UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

DESARROLLO E IMPLEMENTACIÓN DE REGULADORES DIGITALES PARA CONVERTIDORES CONMUTADOS UTILIZANDO MICROCONTROLADORES

Alejandro Castro Beltrán

Tutor: Fernando Jesús López Colino

Ponente: Ángel de Castro Martín

Junio 2015

DESARROLLO E IMPLEMENTACIÓN DE REGULADORES DIGITALES PARA CONVERTIDORES CONMUTADOS UTILIZANDO MICROCONTROLADORES

AUTOR: Alejandro Castro Beltrán

TUTOR: Fernando Jesús López Colino

Human Computer Technology Laboratory

Departamento de Tecnología Electrónica y de las Telecomunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio 2015

AGRADECIMIENTOS

La primera persona que quiero mencionar en este trabajo es a mi tutor Fernando. Quiero agradecerle la ayuda y la paciencia que me ha prestado durante este largo año mientras desarrollaba el trabajo, las dudas resueltas y las infinitas correcciones para tratar de presentar este TFG de la mejor forma posible.

Quiero agradecer estos años en la universidad especialmente a mis amigos Álvaro, Lucas, Antonio, Víctor, Diego, Roberto, Raquel… y todos de los que ahora mismo, al escribir estas líneas, no me acuerdo de mencionar pero que han estado ahí siempre e incluyo de la misma forma.

A mis amigos del colegio, a los cuales por fortuna sigo viendo y nos seguimos riendo y contando historias como el primer día. Huberto, Koke, Richi, Javi, Rafa… gracias por seguir ahí.

Por último, a mi familia, pero en especial a las 3 personas más importantes en mi vida. Mi madre, mi padre y mi hermana, que siempre me han estado apoyando en todo momento desde que empecé a estudiar esta carrera. Muchas gracias por todo.

Madrid

Junio 2015

RESUMEN

El objetivo de este trabajo de fin de grado es desarrollar, mediante las herramientas de software y hardware necesarias, un sistema de control digital para convertidores conmutados para controlar su salida manteniendo una tensión determinada y una dinámica adecuada.

Para lograr este objetivo, se utilizó un microcontrolador especializado en control digital en el que se ha desarrollado un código para calcular, generar y llevar todas las señales de control necesarias hacia el conmutador, a la vez que se lee en tiempo real la tensión de salida. Estas acciones son necesarias para llevar a cabo el control en lazo cerrado del conmutador.

Una vez completada la programación del microcontrolador se conectó a un convertidor conmutado reductor ya construido, para comprobar en un entorno real que el control diseñado funciona correctamente. Estas pruebas han demostrado la validez de la implementación del regulador en el microcontrolador.

PALABRAS CLAVE

Microcontrolador, Control digital, Convertidor reductor DC/DC, PWM, Fuente de alimentación.

ABSTRACT

The objective of this project is to develop, using the software and hardware tools available, a digital control system for a DC/DC Buck converter. The main purpose is to control the output of the converter, maintaining the correct voltage and a good dynamic.

In order to accomplish the objective, a microcontroller specialized on digital processing was used. Later on, it needed to be programmed to be able to calculate, to generate and to bring all the signals to the converter. At the same time, the system output was monitored. These actions are necessary to complete the closed loop control model.

Once the programming phase ended, the microcontroller was connected with the Buck converter and the complete system was measured to make sure it performed without any kind of problem in a real environment. These measures have proved that the implementation was successful.

KEY WORDS

Microcontroller, Digital Control, Buck Converter DC/DC, Power Width Modulation, Power Supply

ÍNDICE GENERAL

ÍNDICE DE CÓDIGO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

En este trabajo de fin de grado se ha realizado el desarrollo de un sistema de control digital de un convertidor conmutado mediante un microcontrolador. En este primer apartado se van a explicar los conceptos generales de la ingeniería de control, haciendo hincapié en lo relacionado con el control digital, así como las posibles utilidades y usos que se le puede dar en un entorno real.

Un sistema de control puede ser de tiempo continuo, en cuyo caso su modelo puede ser representado con ecuaciones diferenciales, o de tiempo discreto, que se representa mediante ecuaciones en diferencias. Cuando es necesario trabajar con éstas últimas (en el control digital con microcontroladores o FPGA sólo se puede trabajar con sistemas discretos), los sistemas en tiempo continuo se discretizan con convertidores analógico a digital.

Asimismo, existen dos formas diferentes de controlar un sistema, llamadas en lazo abierto y lazo cerrado. Con lazo abierto, el control modifica la actuación sobre la planta en función de un patrón determinado. En estos sistemas no se mide la salida para comprobar que todo funcione correctamente, y por lo tanto son de limitada utilidad, por lo general, en aplicaciones de control reales. En lazo abierto sólo se tiene en cuenta un modelo teórico, que se aplica directamente. El otro tipo de sistema se llama en lazo cerrado. La diferencia con respecto al lazo abierto radica en que el lazo cerrado sí considera la salida del sistema, con lo que se dispone de información real sobre cómo de cerca o lejos están los resultados de donde se esperaba estar. Se compara la entrada con la salida, generando una señal resultante de error que le dice al controlador si debe mantener la acción que estaba usando o por el contrario aumentar o reducir la variable de trabajo para acercarse a los resultados esperados.

Ahora que se ha presentado qué es un sistema de control y cómo funciona de una forma general, se va a ahondar más en los detalles del control digital, qué cambios hay con respecto a un sistema de control analógico y las ventajas o inconvenientes que tienen los microcontroladores con respecto a otros dispositivos digitales cuando hablamos de control digital.

1.1 El control digital

El control digital tiene una principal característica que lo diferencia del control analógico, y es que opera en instantes de tiempo determinados, con lo que es un

sistema discreto. Este tipo de sistemas quedan totalmente caracterizados por una ecuación en diferencias.

A la hora de diseñar un sistema de control digital hay que tener en cuenta, por tanto, posibles componentes que serán necesarios para el procesamiento: el primero es un convertidor analógico digital (ADC). Dado que la mayoría de sistemas son continuos (en este mismo Trabajo de Fin de Grado se controla la tensión, que es algo continuo), es necesario un ADC para que el controlador, que es digital (en este caso, un microcontrolador) pueda procesar la señal y aplique el modelo con la ecuación en diferencias.

Además, es necesario tener en cuenta que si el sistema con el que se trabaja es continuo, después del procesamiento de la señal será necesario otro componente: un convertidor digital a analógico (DAC). La señal resultante digital se convierte en analógica para ser la entrada del sistema a controlar. Este sistema recibe el nombre de planta.

El control digital es muy usado en muchas áreas de trabajo de ingeniería, y es básico para el correcto funcionamiento de muchos dispositivos que usamos en la vida cotidiana. Por ejemplo, en la ingeniería marítima se usa para controlar el nivel de agua que hay en las escotillas de un puerto o canal, en ingeniería aeroespacial se usa para controlar correctamente el ángulo de visión de un satélite, en la industria se usa para controlar el correcto funcionamiento de las máquinas de producción, etc. Hay cientos de usos prácticos hoy en día para el control digital [1].

Por lo general, uno podría pensar que usar control digital frente al analógico tiene siempre más ventaja, pero realmente depende de la aplicación. Por ejemplo, los sistemas de control analógicos consiguen un ancho de banda mayor.

Por el contrario, el control digital tiene ciertas ventajas que hacen que hoy en día se utilice en mayor medida. La primera es la duración de los componentes: los componentes digitales son menos susceptibles al medio al que están expuestos, mientras que los analógicos trabajan correctamente a unas condiciones muy específicas. Esto hace que a largo plazo compense el control digital. Otra ventaja importante es que el ruido o las interferencias en los controladores digitales son siempre menores. Suelen estar más protegidos con componentes internos. Finalmente, la ventaja que puede tener más peso es que son fácilmente programables: mientras que en un sistema analógico el cambio en el control requiere un cambio en los componentes, en el control digital basta con reprogramar el controlador.

En contraste con el analógico, el control digital viene limitado en velocidad por la capacidad de procesamiento del controlador (microcontrolador o FPGA, por ejemplo) y que la resolución es limitada: se suele perder precisión al pasar la variable de estudio por los convertidores analógico a digital y viceversa. Esto es debido a que en control digital el sistema numérico para procesar todas las variables es el binario, cuya resolución viene limitada por el número de bits de cada palabra.

1.2 Dispositivos digitales: FPGA y microcontrolador

El controlador que se va a usar en este trabajo de fin de grado es un microcontrolador (µC). Existe otra opción diferente que también es muy utilizada, la FPGA, por lo que se va a detallar un poco las características y diferencias entre ambos dispositivos.

Lo primero en lo que se puede describir una FPGA (figura 1) con respecto a un µC (figura 2) es la forma de programar cada componente. Mientras que éste último casi siempre utiliza lenguaje C, la FPGA se suele programar con lenguaje HDL. La principal diferencia entre ambos radica en que mientras que en C se programan instrucciones de manera secuencial, en HDL se describen los componentes con modelos top-level y cómo funcionan de una manera diferente.

Otra diferencia sustancial es la naturaleza del propio componente. El µC está limitado en tiempo, pero ya dispone de módulos pre-programados como un ADC o un PMW de fácil acceso, mientras que en la FPGA suele ser necesario utilizar HDL para diseñar estos módulos. A cambio, en una FPGA se puede ejecutar todo en un mismo instante de tiempo, ya que lo que limita es el espacio.

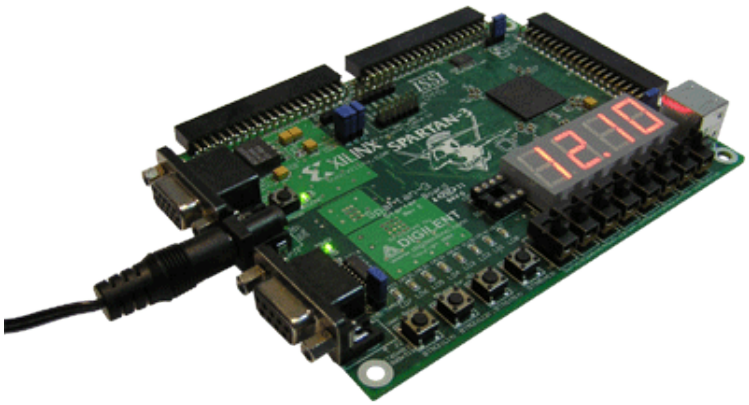


Figura 1: FPGA

Figura 2: Microcontrolador

1.3 Motivación del trabajo

La finalidad de este Trabajo de Fin de Grado es realizar una aplicación de control digital para un convertidor conmutado. Se construyó recientemente un convertidor multifase en el laboratorio, por lo que se vio una gran oportunidad de realizar una aplicación de control real con un dispositivo digital.

Se ha decidido utilizar un microcontrolador para realizar este Trabajo de Fin de Grado por encima de una FPGA por dos razones principales, primero por el tipo de programación, y segundo porque se quería aprovechar los módulos (PWM, timer, ADC) de los que dispone el componente para realizar un control más eficaz con los recursos que dispone. Además, el microcontrolador que se ha elegido para trabajar (el modelo TMS320F2808 de Texas Instruments) cuenta con un entorno de trabajo intuitivo y fácilmente programable.

El control que se quiere hacer en el trabajo es de lazo cerrado, midiendo en todo momento la salida y aplicando la ecuación en diferencias a una frecuencia que vendrá determinada por la planta (el convertidor conmutado).

1.4 Estructura de la memoria

- En el capítulo 2 se va a hacer una recapitulación de estado del arte, explicando de forma general qué avances se han hecho en el campo del control digital.

- En el capítulo 3 se exponen las principales características del convertidor que se ha utilizado en este trabajo.

- En el capítulo 4 se describe el proceso de la inicialización de los módulos del microcontrolador que se usarán en la aplicación de control final.

- En el capítulo 5 se programa la aplicación de control. También se explican los problemas de diseño que se han encontrado y cómo se han terminado afrontando.

- En el capítulo 6 se lleva a cabo la conexión del microcontrolador y la fuente conmutada, se explican en detalle las conexiones, así como las modificaciones realizadas para adaptar la salida del convertidor al µC.

- En el capítulo 7 se realizan las pruebas experimentales con varios reguladores y diferentes cargas, y se comentan los resultados obtenidos.

- Finalmente en el capítulo 8 se exponen las conclusiones del proyecto, y se comentan posibles mejoras futuras de cara a profundizar más este trabajo.

2. ESTADO DEL ARTE

La presencia del microcontrolador como hardware de control en los últimos años ha experimentado un auge muy importante. Son muchas las aplicaciones que pueden ejecutarse hoy en día con un micro especializado en procesamiento digital. Las prestaciones que ofrece suelen ser mayores a las de un dispositivo de carácter más general, por lo que su demanda, y como consecuencia la disponibilidad de diferentes modelos, se ha incrementado.

Las aplicaciones de control en las que se puede utilizar un microcontrolador son de varios tipos. A continuación se explican algunos ejemplos de uso en un entorno real.

Se puede utilizar un µC para controlar sistemas electrónicos de jardinería vertical [2], que monitorizan la cantidad de luz que debe llegar a cada planta o vivero en un determinado momento. En este sistema no se prioriza la velocidad de procesamiento, sino la exactitud de la medida y el guardado de datos para su posterior análisis (se suele necesitar una cantidad de luz y agua distinta en cada plantación). Para ello se suelen buscar procesadores con determinadas características; por ejemplo en la referencia [2] se utiliza el modelo PIC18F4520 que tiene la memoria Flash y SARAM más grande su familia, lo que permite guardar el doble de medidas que otros micros de las mismas características.

En el campo doméstico, es notable el uso de µC para controlar el correcto uso de los electrodomésticos. En la referencia [3] se investiga una forma de controlar el motor de una lavadora mediante una aplicación que permita establecer un número máximo de centrifugaciones o la apertura de las válvulas de drenado en un determinado momento. Es importante que el microcontrolador en este caso priorice la rapidez en la actuación, ya que el motor del electrodoméstico depende directamente del micro para decirle las revoluciones por minuto y cuánto tiempo debe hacerlo. Para evitar averías, estos sistemas cuentan con un sistema de protección que hace, por ejemplo, que si la lectura del sensor está en un rango incorrecto o no se lee nada durante unos segundos, el motor se pare.

Uso de un µC para controlar la altura, presión ejercida en el sistema y velocidad de movimiento de un ascensor [4]. Hoy en día se realizan muchas aplicaciones en este campo, debido a la prioridad por mejorar la seguridad de las personas en los ascensores. Los microcontroladores que se usan en este campo suelen ser más complejos y requieren exactitud en la medida (el ascensor no se puede quedar parado unos centímetros debajo o encima de la planta), un control de la presión que el ascensor ejerce en el sistema [4] para en el caso de que se detecte más peso del

permitido el ascensor se bloquee, y además controlar que la velocidad del ascensor, tanto en subida como en bajada, nunca pase de un valor máximo.

En el campo de los convertidores conmutados también se ha experimentado avances, debido principalmente a que un convertidor conmutado pierde menos energía que un regulador lineal, por lo que su uso se ha extendido. Se ha logrado desarrollar convertidores multifase cada vez más sofisticados, que mejoran la eficiencia de forma muy notable. La principal ventaja de los convertidores multifase es un rizado mucho menor en la corriente, y con intensidades que pasan a través de las inductancias de forma mucho más controlada (como se indica en la referencia [5]).

Una de las áreas donde ha ganado mucha importancia el convertidor conmutado es la fotovoltaica. Cada vez más, se ven sistemas de luz de baja potencia basados en tecnología LED, por lo que un convertidor que regule el voltaje que llega a estas células de luz es una opción que se usa e investiga de forma continua [6]. La principal tarea del convertidor en este área es la de controlar (mediante un microcontrolador u otro dispositivo digital) la intensidad de la luz, y es muy común verlos en sistemas de luz regulados manualmente.

Los convertidores se usan en muchas otras áreas de investigación y desarrollo, además de usos más habituales. Una aplicación de control donde se está investigando es, por ejemplo, el uso del convertidor como herramienta para electrocutar con una tensión exacta a reses en ganadería antes de la producción de alimentos, en vez del uso de los métodos tradicionales (muerte en matadero) [7]. La salida de tensión de un convertidor que aumente el voltaje a niveles lo suficientemente altos para provocar la muerte a un animal (cada animal posee su propio umbral). Esto permite reducir el dolor que sufre el animal en los mataderos.

Con el aumento de vehículos eléctricos año a año en las carreteras, el uso de un convertidor conmutado multifase como herramienta de regulación de voltaje en los sistemas del vehículo es una realidad que ya se está incluyendo en algunos modelos comerciales. Se ha logrado desarrollar, utilizando una batería de litio y un convertidor, un sistema que utilice la energía que el coche no usa para recargar la batería eléctrica [8], lo que permitiría aumentar en el futuro la autonomía 50 Km de media antes de volver a tener que conectarlo a la red eléctrica.

En definitiva, el campo del desarrollo e investigación del control digital, mediante microcontroladores u otros dispositivos digitales se encuentra en pleno auge, y es posible ver sistemas que ahora se controlan digitalmente cuando hasta hace poco se

utilizaban métodos analógicos. El uso de los convertidores es muy común en aplicaciones que requieran optimizar el consumo de energía.

3. DESCRIPCIÓN DEL BUCK

El circuito que se va a regular en este trabajo es un convertidor conmutado reductor (Buck). Su función es la de transformar una tensión continua en la entrada en una tensión continua menor a la salida (figura 1, [9]).

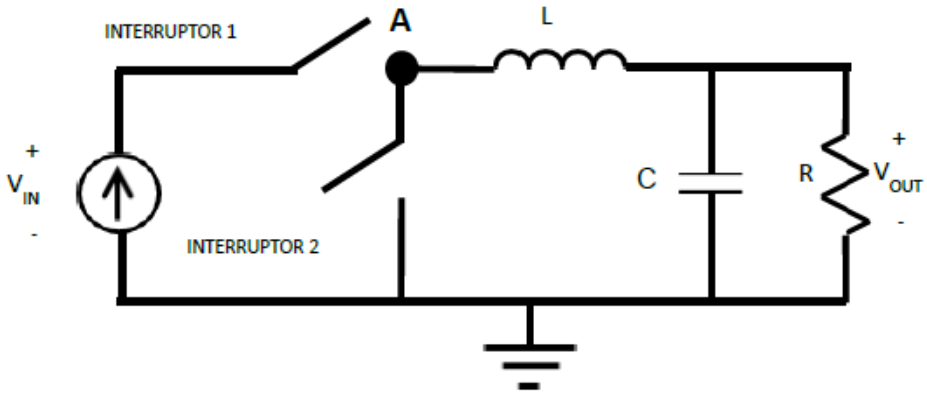


Figura 3: Esquemático de un regulador tipo Buck.

La operativa sobre este circuito es la de cerrar los interruptores complementariamente en la proporción de tiempo adecuada para obtener la tensión de salida adecuada. El control de este circuito debe evitar que ambos interruptores se cierren a la vez para evitar un cortocircuito.

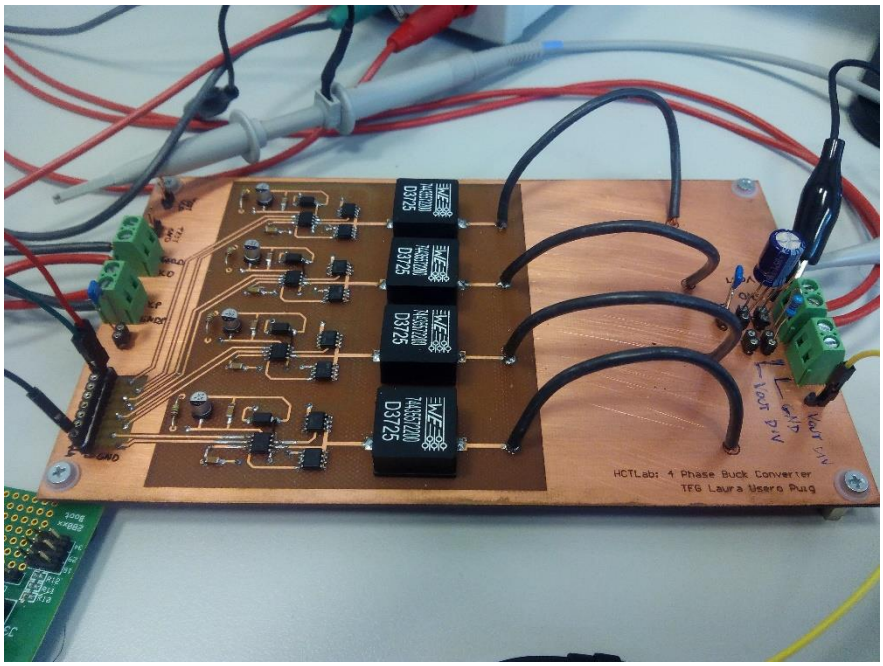


Figura 4: Detalle del regulador conmutado utilizado en el trabajo

En el grupo HCTLab está disponible un Buck multifase [10] que ha sido utilizado para este trabajo (figura 4). Aunque dicho Buck incluye cuatro fases, este trabajo se ha centrado en una de ellas funcionando de manera aislada. Las características del Buck construido son:

- Condensador de **220 µF** común
- Bobinas de **22 µH** por cada fase

Este convertidor se ha probado a frecuencias de conmutación de 100 kHz, 200 kHz y 400 kHz. En este trabajo se realizará el control a 100 kHz.

La carga o resistencia del final del esquemático no está incluida en la propia placa, lo que va a permitir utilizar cargas que puedan ir cambiando durante las diferentes pruebas. Para ello se ha utilizado una panel de resistencias en paralelo con interruptores que permite ir cambiando la carga según qué interruptores estén abiertos o cerrados (figura 5).

Figura 5: Resistencias que actúan como carga variable en el control

Las tres primeras resistencias de la parte superior son resistencias de 2 Ω. Las dos siguientes son de 5 Ω, las dos naranjas más pequeñas tienen un valor de 10 Ω y finalmente la que no tiene interruptor es de 100 Ω. La tabla 1 incluye algunas de las configuraciones posibles.

INTERRUPTORES ACTIVADOS	RESISTENCIA EQUIVALENTE
Ningún interruptor activado	100 Ω
Interruptor 1	9,091 Ω
Interruptor 3	4,762 Ω
Interruptor 5	1,961 Ω
Interruptores 1 y 3	3,226 Ω
Interruptores 1 y 4	1,639 Ω
Todos	0,474 Ω

Tabla 1: Posibles combinaciones de la carga total al final del reductor

El convertidor cuenta a su salida con la posibilidad de incluir en unos pines un divisor de tensión. Este divisor se ha utilizado en este proyecto, dado que era necesario para un correcto funcionamiento del ADC. El ADC del micro, debido a sus especificaciones, lee valores desde 0 a 3 voltios, saturando la lectura a un valor digital de 4096 para todo aquello que esté por encima de 3 voltios.

El divisor se ha realizado de tal forma que si la salida llega a 12 V, el ADC tenga a su entrada 2,97 V. Para ello las resistencias empleadas en el divisor son de 3 kΩ y 9,09 kΩ. La ecuación del modelo final queda (ecuación 1):

Ecuación 1: $$V_{adc} = \frac{Vout \cdot 3\ k\Omega}{12{,}09\ k\Omega}$$

Los pines del regulador dan la opción de incluir un condensador que reduzca el ruido producido por las resistencias utilizadas en el divisor, pero se optado por no incluirlo.

Para alimentar al conmutador se ha utilizado la fuente de voltaje EA-PS 2342-06 B (figura 6) que provee de una tensión de 12 V a la placa (alimentación analógica) y 12 V para la alimentación digital. Según las especificaciones de [10] se puede alimentar la tensión digital con un valor entre 10 y 20 V, por lo que se ha optado por un valor de 12 V.

Figura 6: Detalle de la fuente de potencia utilizada en las pruebas de control

4. DESCRIPCIÓN DEL MICROCONTROLADOR

Antes de explicar el proceso de desarrollo que se ha seguido durante este Trabajo de Fin de Grado, es necesario definir las principales características que tiene el microcontrolador elegido para el proyecto.

4.1 El microcontrolador TMS320F2808

Durante las fases preliminares del trabajo se ha decidido que se iba a utilizar un microcontrolador por las diferentes razones expuestas anteriormente. El primer paso antes de comenzar el desarrollo de la aplicación de control es tener en cuenta qué modelo se va a usar, en qué entorno de trabajo se va a programar y qué tipo de lenguaje se va a utilizar. El resultado de estos estudios fue el microcontrolador TMS320F2808 de Texas Instruments [11] (figura 7).

La principal ventaja de este µC es la fácil programación de sus módulos, que vienen preparados para soportar aplicaciones no muy exhaustivas pero lo suficientemente rápidas para investigar o experimentar. De hecho, el kit utilizado es el "Experimenter's Kit" que trae una conexión por USB y un entorno de trabajo intuitivo, cuyo lenguaje de programación es C.

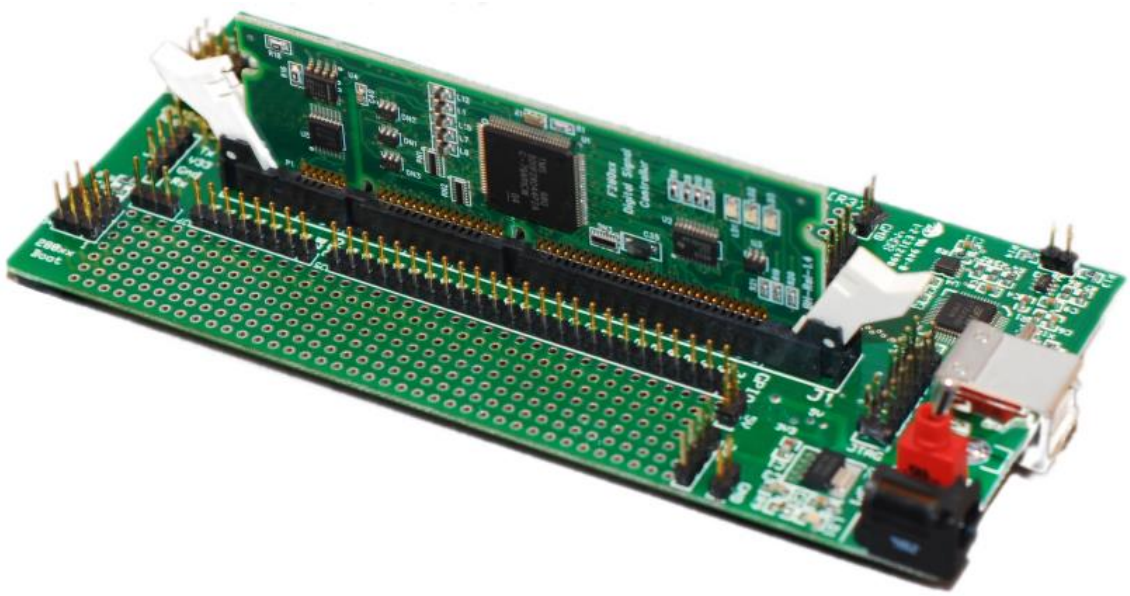


Figura 7: Microcontrolador TMS320F2808 Experimenter's Kit

Las características técnicas generales del microcontrolador elegido son las siguientes [11]:

- Frecuencia del reloj del microcontrolador de 100 MHz.
- CPU de 32 bits.
- Memoria flash de 32Kx16 bits y una memoria SARAM de 18Kx16.
- Bloque de interrupciones que soporta interrupciones de todos sus componentes y módulos.
- 3 timers en la CPU de 32 bits.
- 16 módulos PWM, todos ellos con comparadores de alta resolución.
- ADC de 12bits y 16 canales que pueden leer muestras de forma simultánea.
- 35 puertos GPIO de entrada/salida.
- Pestañas con conexiones a VCC de 3,3 V y 5 V.

Con estos datos técnicos se puede asegurar que el µC elegido cumple con los mínimos necesarios para realizar una aplicación de control del tipo que se busca, ya que cuenta con una velocidad que a priori será suficiente, una CPU de 32 bits, módulos PWM que serán necesarios para generar la señal de actuación, módulo ADC con lecturas simultáneas, suficientes puertos GPIO para llevar y traer señales del micro al conmutador y viceversa, y por si hiciese falta también se cuenta con un timer.

El entorno de trabajo con el que se trabaja con el TMS320F2808 es el programa de Texas Instruments llamado Code Composer Studio. Este programa se provee con el kit comprado con el microcontrolador y tiene una interfaz fácil de usar. El lenguaje de programación es C, y soporta el acceso a todos los registros y posiciones de memoria del micro con las respectivas instrucciones (figura 8).

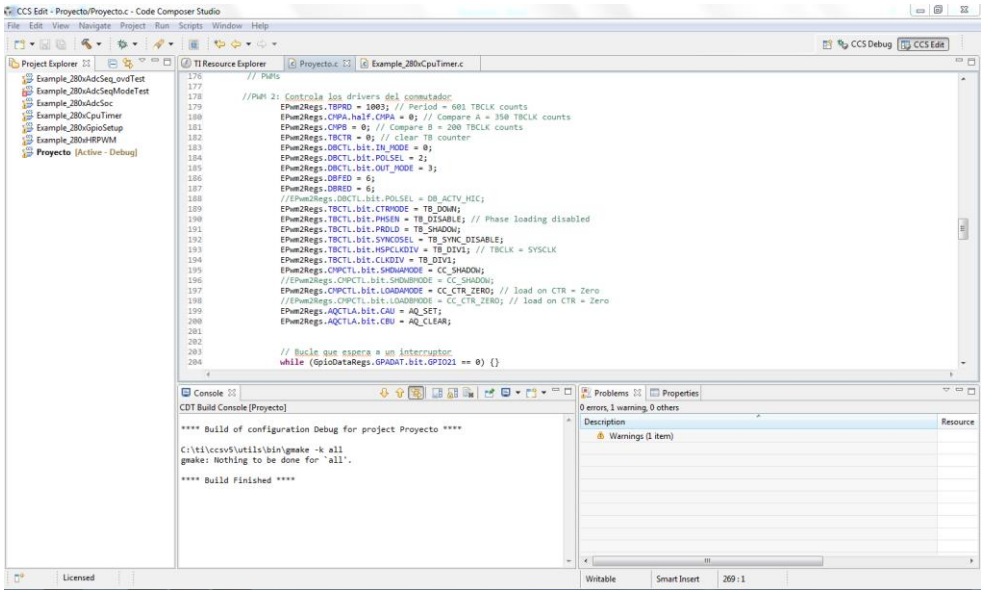


Figura 8: Interfaz de programación de Code Composer Studio

Con Code Composer Studio se puede depurar también en tiempo real dónde se guardan las variables del programa, poner breakpoints para pausar en un determinado punto la ejecución y modificar en mitad de una ejecución determinados valores cómo las variables globales y los registros del µC.

Otra de las ventajas de trabajar con este entorno de trabajo es el hecho de poder cambiar de una forma sencilla el tipo de conexión que se usa con un emulador, de tal forma que si en el futuro se quisiese cambiar el microcontrolador por otro modelo más moderno o con mejores prestaciones, por ejemplo, podría utilizarse el mismo código con sólo unos pocos cambios en el compilador.

4.2 El lazo cerrado

Antes de comenzar a programar es conveniente explicar de una forma más detallada qué tipo de control se va a utilizar específicamente en este trabajo. La principal premisa del control es hacerlo dinámico y qué responda a variaciones del sistema en tiempo real, así pues es necesaria la aplicación de un modelo en lazo cerrado.

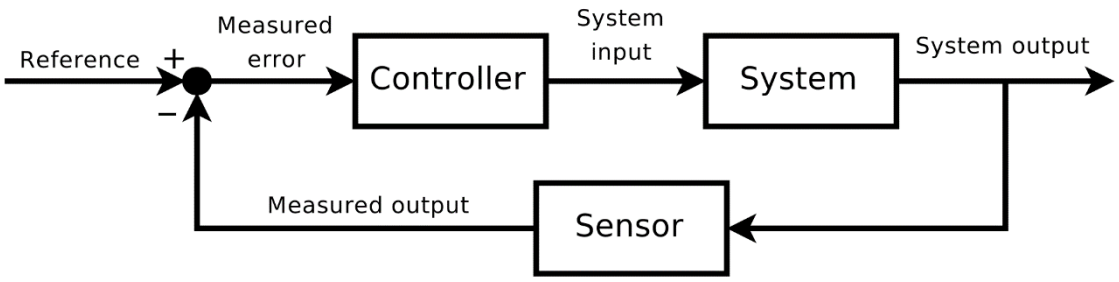


Figura 9: Esquemático general de un sistema de control en lazo cerrado

En un lazo cerrado la entrada del sistema, llamada consigna o referencia, es el valor que la aplicación de control quiere alcanzar. La salida final del sistema es la que se compara con la consigna o referencia, generando una señal de error. La señal de error se envía al controlador, que atendiendo al valor que posee dicha señal genera la actuación correspondiente (figura 9, System Input) que se introduce en la planta. La planta generará la salida que se utilizará para compararla con la referencia de entrada, comenzando de nuevo un nuevo ciclo.

En el sistema desarrollado en este trabajo, el primer esquemático puede desarrollarse de una manera más específica en la figura 10:

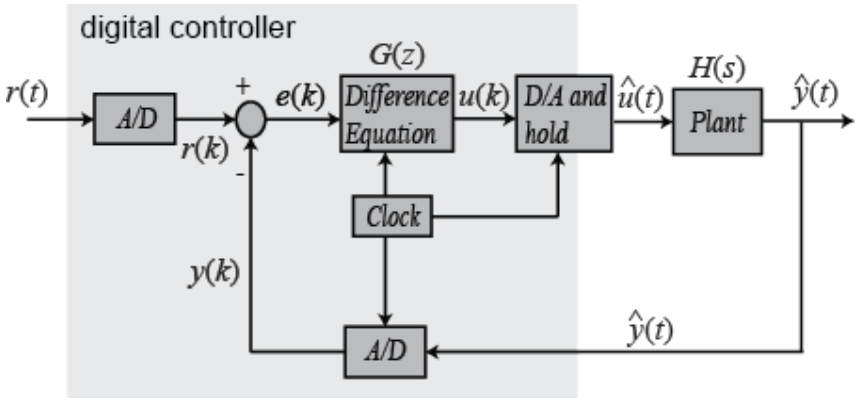


Figura 10: Esquemático detalle de un sistema de control digital en lazo cerrado

Este segundo esquemático (figura 10, referencia [12]) responde a un caso más particular en las aplicaciones de control digital, y se corresponde con el micro utilizado en este Trabajo de Fin de Grado. Dado que el microcontrolador sólo trabaja con sistemas discretizados (ecuaciones en diferencias), es necesario que la variable de trabajo usada (voltios, variable continua) se discretice antes de poder ser

procesada por el microcontrolador. Para llevar a cabo esta tarea, el µC cuenta con un módulo ADC (A/D) con el que transforma las variables analógicas en valores digitales que sí pueden ser leídos por el micro.

Una vez discretizada la señal de error, se lleva a cabo el propio algoritmo de control, que viene representado por una ecuación en diferencias [13] (ecuación 2).

$$Ecuación\ 2: \qquad a_0 \cdot y[k] = \sum_{i=0}^{N-1} b_i \cdot x[k-i] - \sum_{j=1}^{N-1} a_j \cdot y[k-j]$$

Las ecuaciones en diferencias son la representación matemática de un modelo formal y teórico, cuyos coeficientes b_i y a_i son fijos y vienen dados por el modelo aplicado, mientras que los vectores x e y representan, respectivamente, los valores de error y actuación en un instante de tiempo k. Por ejemplo, en un sistema de segundo orden, el valor x[k-2] estará referido a la señal de error en dos instantes de tiempo pasadas. El resultado y[k] es la propia actuación en el instante actual, que será la entrada a la planta. En el caso del convertidor conmutado es un ciclo de trabajo.

Un PWM es una modulación por ancho de pulso que genera una señal periódica cuadrada, siendo el ciclo de trabajo el porcentaje de tiempo que la señal está activa. Así pues, si se obtuviese un valor y[k] sobre 1 de 0.6, el módulo PWM generaría una señal periódica en el que el 60 % del tiempo estuviese en ‘1’, y el 40 % restante a ‘0’. Finalmente la señal de salida del convertidor se volvería a pasar por el ADC del controlador digital, comenzando un nuevo ciclo de control.

4.3 Programación de los módulos del micro

Se ha procedido a comenzar el desarrollo de la aplicación de control, que será la parte más importante del proyecto. Para ello se ha dividido el desarrollo en varias etapas, comenzando por la inicialización y configuración de los módulos del microcontrolador [14].

Las principales características de la aplicación son: que se realice un control en base a un modelo teórico, cuyos coeficientes y valores se sacarán de un regulador teórico desarrollado más adelante; que la aplicación funcione a 100 kHz (esto es crítico ya que la fuente conmutada que vamos a controlar funciona a 100 kHz), y que el control sea programable de una forma sencilla.

4.3.1 Programación del ADC

La primera etapa del desarrollo comenzó con una primera configuración de los módulos del micro que se iban a utilizar en el proyecto. Lo primero que se programó es el ADC; se ha realizado la configuración del módulo en base a la necesidad del programa. En principio sólo se va a necesitar leer un dato (la tensión de salida de convertidor conmutad), por tanto, una configuración del ADC en modo lectura simple bastará para el propósito del proyecto por el momento (código 1).

```
1. AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK;
2. AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS;
3. AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;
4. AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;
5. AdcRegs.ADCTRL1.bit.CONT_RUN = 1;
6. AdcRegs.ADCTRL2.all = 0x2000;
```

Código 1: Configuración del ADC en modo lectura simple

La dos primeras líneas se refieren a que se va a utilizar el reloj interno del ADC, y que lo hará con la misma frecuencia que el reloj general. La tercera indica que se va a leer en modo cascada (es decir, no es una lectura simultánea de varios canales), seguidamente se indica que se va a leer el registro 0 del ADC, y finalmente en la línea 5 se indica con un flag que el ADC debe estar leyendo en modo secuencia continuamente. La última línea activa el ADC, que empieza a leer desde ese punto.

Con esta primera configuración se ha procedido a probar el funcionamiento del ADC con unas pruebas sencillas, que han permitido ver que el rango de las lecturas se sitúa entre 0 y 3 voltios. Valores por debajo de 0 son leídos simplemente como un 0 digital, mientras que valores a partir de 3 V saturan siempre a 4095, con lo que se confirma la resolución de 12 bits del ADC ($2^{12} = 4096$).

4.3.2 Programación del PWM

El siguiente módulo que sobre el que se ha trabajado es el módulo PWM del microcontrolador. Para ello se ha realizado una primera configuración con un PWM estándar [15] que funcione a 100 kHz (código 2), que será la frecuencia que se busque en el programa, tal y cómo se ha matizado anteriormente.

```
1. EPwm2Regs.TBPRD = 1000;
2. EPwm2Regs.CMPA.half.CMPA = 400;
3. EPwm2Regs.CMPB = 600;
4. EPwm2Regs.TBCTR = 0;
5. EPwm2Regs.TBCTL.bit.CTRMODE = TB_DOWN;
6. EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;
7. EPwm2Regs.AQCTLA.bit.CBU = AQ_CLEAR;
```

Código 2: Inicialización del PWM

Con el reloj interno del microcontrolador sin modificar, es decir, que funcione al 100 % de la velocidad, el número de ciclos de PWM que son necesarios para alcanzar una frecuencia de 100 kHz es de 1000. Será este pues el número de ciclos que representará el 100 % del ciclo de trabajo en esta aplicación. Cada módulo PWM dispone de dos comparadores, A y B (figura 11), que se usan para señalizar el momento en el que el PWM debe subir o bajar. Finalmente en las dos últimas líneas del código se especifica qué acción se tomará cuando se cumpla la condición o el número de ciclos de un comparador: en este caso, y como inicialización, cada vez que se llegue al valor del comparador A se hace un SET (poner a 1) y al llegar al comparador B se hace CLEAR (poner a 0).

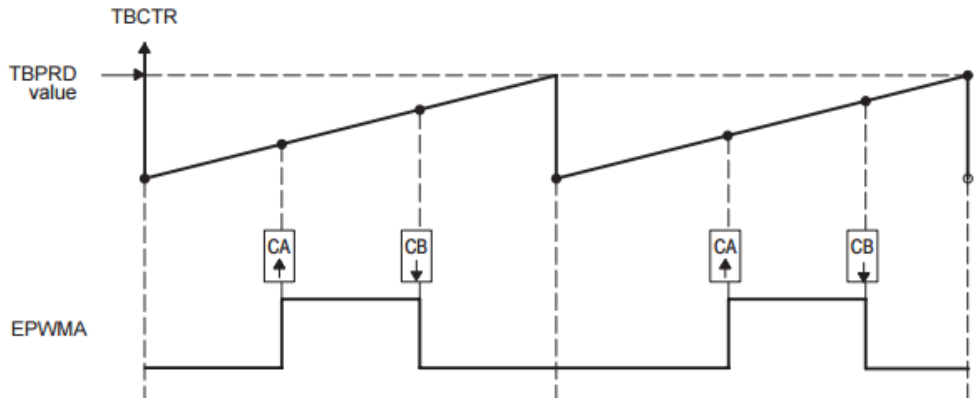


Figura 11: Inicialización estándar del módulo ePWM

Hay una serie de registros en el PWM que sirven para indicar si se quiere utilizar alguna de las funciones extra de las que dispone el PWM (enhanced PWM). Es necesario inicializarlos para que el PWM funcione correctamente (código 3), aunque se han dejado todos ellos en sus valores por defecto (desactivados), ya que no tienen relevancia en este trabajo.

```
// Registros de control que no se utilizan (valores por defecto)

1. EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;
2. EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;
3. EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
4. EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
5. EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
6. EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
7. EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
```

Código 3: Registros de control del PWM por defecto

Una de las funcionalidades extra de las que dispone el módulo PWM es el hecho de poder configurarlo como un PWM de alta resolución. Esto a priori está pensado para frecuencias de trabajo de más de 200 kHz, llegando a ser necesario en frecuencias cercanas a 1 MHz. Dado que en la aplicación de control se trabajará a 100 kHz, no se ha realizado la configuración del PWM en alta resolución, aunque es interesante tenerlo en cuenta para futuras mejoras del programa si se quisiese aumentar la frecuencia.

4.3.3 Programación del timer

El tercer módulo que se ha inicializado como parte preliminar del desarrollo es uno de los timers disponibles de la CPU del micro. Este timer será, en un principio, el que se utilizará como llamada para que el modelo de control con la ecuación en diferencias se actualice.

La configuración [16] se realiza de tal forma que se genere una interrupción por timer periódica. Para ello se activa la tabla de interrupciones correspondiente y se genera el timer con los requisitos que necesite el programa (código 4).

```
1. PieVectTable.TINT0 = &cpu_timer0_isr;
2. InitCpuTimers();
3. PieCtrlRegs.PIEIER1.bit.INTx7=1;
4. EINT;
5. ERTM;
6. ConfigCpuTimer(&CpuTimer0, 100, 10);
7. StartCpuTimer0();
```

Código 4: Inicialización del timer de la CPU

La primera línea del código hace que cada vez que haya una interrupción el código salte a la función cpu_timer0_isr. La segunda línea llama a la función interna del micro que inicializa los timers de la CPU. Seguidamente se activa la tabla de interrupciones siete que es la que contiene a la interrupción del timer (línea 3), mientras que los comandos EINT y ERTM se refieren, respectivamente, a "Enable Interrupt" y "Enable Global Real Time Interrupt". Finalmente (línea 6) se configura el timer pasándole como parámetros la frecuencia en MHz (utilizamos la misma frecuencia que el reloj del micro, 100 MHz), y una cifra en microsegundos que será a la frecuencia a la que se producirá la interrupción. Dado que el programa tiene que realizarse a 100 kHz, las interrupciones deben producirse cada 10 µs (tercer parámetro). La última línea activa el timer desde ese mismo instante.

4.3.4 Configuración del puerto GPIO

Para finalizar la implementación de los módulos del micro, se ha realizado también una primera prueba de un puerto GPIO que funciona como puerto de salida. En esta configuración, se utilizarán los GPIOS como señales de depuración y medida.

```
1. GpioCtrlRegs.GPADIR.bit.GPIO20 = 1;
2. GpioCtrlRegs.GPAPUD.bit.GPIO20 = 0;
3. GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0;
```

Código 5: Inicialización de un puerto GPIO de salida

Las 3 líneas especifican, respectivamente, que el puerto va a ser de salida, que va a ser configurado como pull-up, y que el puerto funcionará como un puerto de salida estándar.

5. IMPLEMENTACIÓN DEL REGULADOR

5.1 Primera versión del regulador

Para realizar el programa de control se necesitará, en primer lugar, leer el dato de la salida de la planta desde el ADC. Este dato se comparará con una consigna, generando una señal de error. Esta señal de error será la que se utilice, junto a los coeficientes de un modelo teórico, para obtener el resultado de la ecuación en diferencias, que no es otro que la actuación que hay que aplicar a la planta, que en será el ciclo de trabajo de los PWM conectados a los interruptores.

El programa entonces funcionará con un bucle infinito durante toda la ejecución del programa, que lea el canal respectivo del ADC que se ha configurado previamente (en este caso, el 0), y guardar el resultado leído en uno de los registros resultado del ADC (se ha asignado ADCRESULT0) (código 5).

```
1.  for(;;)
2.  {
3.        while (AdcRegs.ADCST.bit.INT_SEQ1== 0) {}
4.        AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;
5.        leidoADC1 = ((AdcRegs.ADCRESULT0>>4));
6.        // GPIO que cambia de valor al iniciar la ejecución
7.        GpioDataRegs.GPADAT.bit.GPIO20= 1;
8.        // Señal de error
9.        x0=consignaPrueba-leidoADC1;
10.       // Ecuación en diferencias. transformación a ciclo de trabajo
11.       y0 = ((b0*x0)+(b1*x1)+(b2*x2))-((a1*y1)+(a2*y2));
12.       y0 = y0 * 1000;
13.       // Saturación de máximo y mínimo
14.       if (y0 > 1000) y0 = 1000;
15.       else if (y0 < 0) y0 = 0;
16.       // Desplazamos en el tiempo
17.       y2=y1; y1=y0; x2=x1; x1=x0;
18.       // Se le pasa al PWM el ciclo de trabajo
19.       EPwm2Regs.CMPB = y0;
20.       // GPIO que cambia de valor al terminar la ejecución
21.       GpioDataRegs.GPADAT.bit.GPIO20= 0;
22.       // Delay que deja el programa funcionando a 100KHz
23.       for (i=0; i<30; i++) {}
    }
```

Código 5: Bucle infinito que realiza el programa de control.

El bucle while que hay en la línea 3 sirve para esperar a que haya una interrupción del ADC. Que se produzca una interrupción da la señal necesaria para saber que ya se puede guardar el dato, que será válido. Tras ello, en la cuarta línea se hace un clear de la interrupción (como un asentimiento) y se procede a guardar el dato leído en una variable declarada manualmente (en este caso, "leídoADC1").

En el código se puede observar cómo el puerto GPIO20 se inicializa a 1 al comenzar con el algoritmo de control (línea 7) y se vuelve a poner a 0 al terminar (línea 21), dando una estimación bastante aproximada del tiempo que lleva al programa realizar cada iteración. Hay que recordar que una de las características del programa es que corra a 100KHz, ni más ni menos. Es por ello que se ha declarado un bucle for al final del bucle principal (línea 23) que actúa como un delay (se ha optado por no incluir el timer en esta primera versión).

La señal x0 se refiere a la señal de error generada en este instante. Une vez obtenida, (línea 9) actualiza la ecuación en diferencias, cuyos coeficientes a0, a1, a2, b0, b1 e b2 vendrán declarados al principio del programa en base a un modelo de regulador formal. Por ahora se han realizado las primeras pruebas con coeficientes aleatorios (que suelen tener valores decimales entre 0 y 1), simplemente para observar el funcionamiento correcto.

Una vez que se ha obtenido la actuación mediante la ecuación en diferencias se va a proceder a convertirla a ciclos de trabajo (línea 19). En un entorno sin cambios, habría que tener en cuenta simplemente que una actuación de 0.3 equivale a 300 ciclos activos sobre 1000.

En la línea 17 se pasa la actuación y el error en este instante (y0 y x0) a las variables de tiempo anterior (y1, x1, etc), de tal forma que en la próxima iteración se trabaje en la ecuación en diferencias con los valores de la última iteración. Al finalizar, se copia el ciclo de trabajo al registro correspondiente del módulo PWM (comparador B en este caso) y se cambia el valor del puerto GPIO20.

5.2 El problema de la velocidad de procesamiento

Con la primera versión del programa se ha procedido a realizar una primera prueba con coeficientes aleatorios pero representativos de los coeficientes que luego se utilizarán en el modelo formal (números decimales que pueden estar entre 0 y 1). El primer problema al que se ha hecho frente es el tiempo de ejecución del programa: no se ha podido llegar a 100 kHz (figura 12).

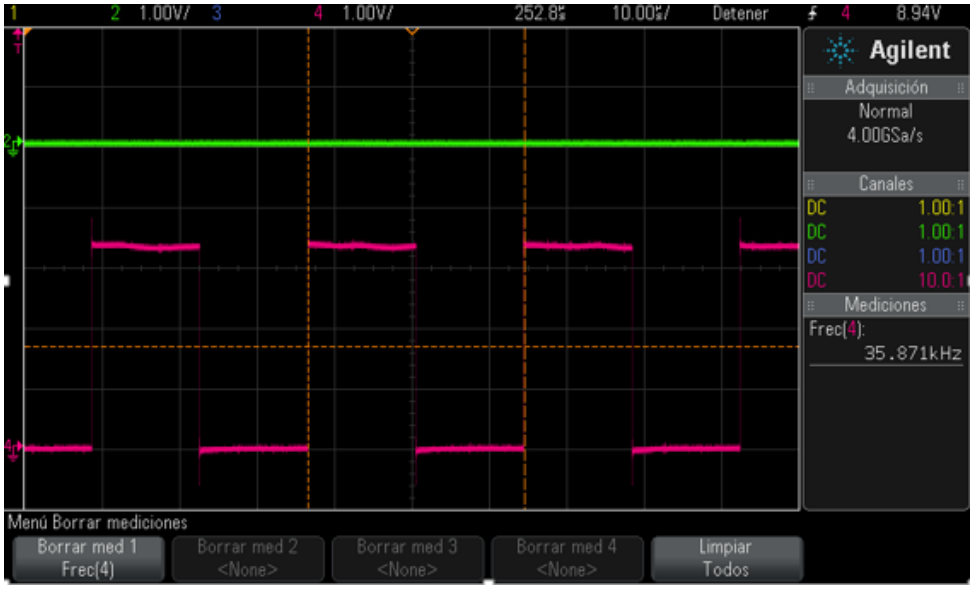


Figura 12: Primera prueba con coeficientes decimales

Este problema se ha producido por dos razones: La primera, y más importante es que se está trabajando con coeficientes de tipo float (coma flotante) que son números decimales, y cuyo procesamiento por parte del microprocesador es muy lento. Si a eso le añadimos que estos coeficientes se multiplican varias veces durante la ecuación en diferencias, llegamos a un efecto en cadena que hace que el programa funcione alrededor de 70 kHz (35 kHz · 2 ya que cada ejecución se cambia el valor del GPIO).

El segundo problema que está agravando la situación es el hecho de que se están multiplicando y dividiendo números que están en la memoria principal del programa, y no en los registros. Aunque esto no suele ser algo crítico, sí que empeora un poco la velocidad de procesamiento.

Para resolver esto se ha intentado realizar una serie de pasos que redujeran la carga de trabajo del micro, de tal forma se optimizara el código realizado. En un principio se han explorado dos cambios que afectan de forma clara a una mejora en la velocidad. El primero ha sido intentar forzar que los coeficientes del modelo formal se guarden en registros del micro, ya que debería ser más rápido cuando el procesador necesite acceder a ellos (con respecto a estar guardados en memoria). Para ello sólo se necesita poner register delante de los coeficientes que se quieren guardar en registros y el compilador debería forzarlos a guardarse ahí (código 6).

El segundo cambio consistió en cambiar en las opciones del compilador las opciones necesarias [17] para que siempre se priorice por encima de todo la velocidad antes que cualquier otra cosa (figura 13).

```
1. // Coeficientes a cambiar manualmente segun modelo teórico
2. Register int b0=0.55;
3. Register int b1=-0.3;
4. Register int b2=0.25;
5. Register int a1=-0.8;
6. Register int a2=0;

7. // Vector de actuación en los instantes t0, t1 y t2
8. Register long y0=0;
9. Register long y1=0;
10. Register long y2=0;
11. // Vector de error en los instantes t0, t1 y t2
12. Register long x0=0;
13. Register long x1=0;
14. Register long x2=0;
```

Código 6: Coeficientes del modelo teórico puestos como registros

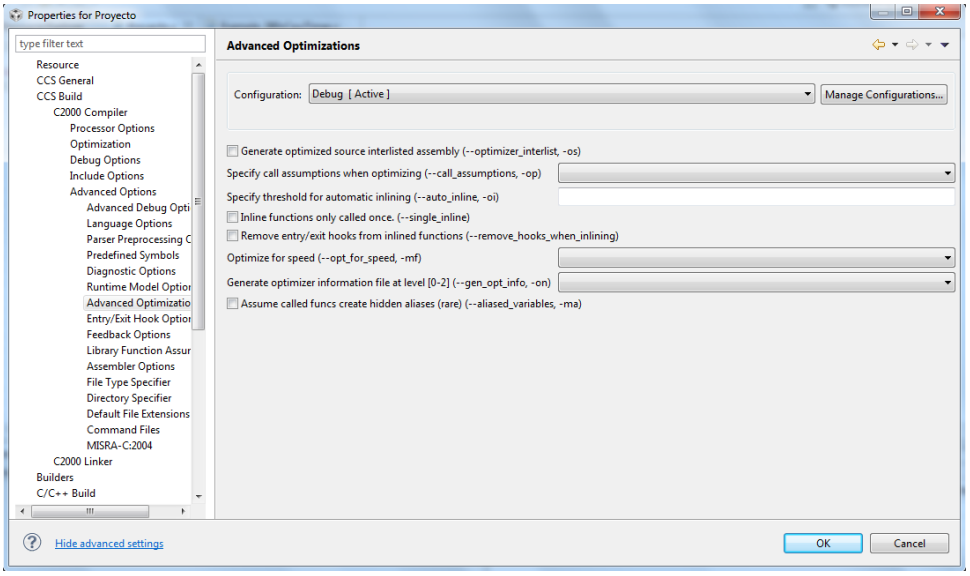


Figura 13: Opciones de optimización en el compilador

Los resultados de estos dos cambios no han sido satisfactorios. A pesar de que se ha conseguido aumentar la velocidad hasta 125 kHz (figura 14), se ha descubierto posteriormente que no hay suficientes registros para que el micro pueda guardar todos los coeficientes del modelo formal, así como los vectores de error y actuación (el micro dispone de hasta 8 registros de uso inmediato, mientras que se han declarado 11 variables de programa como registros), por lo que uno se puede

preguntar cómo se ha conseguido tal velocidad si no todos las variables pueden ser guardadas a la vez como registros.

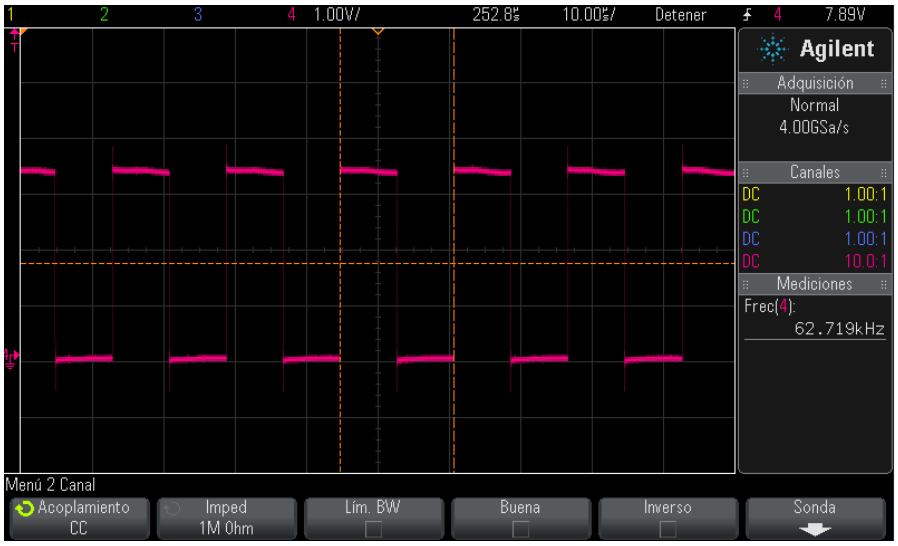


Figura 14: Pruebas con la máxima rapidez en las opciones del compilador. Se alcanza la velocidad necesaria pero se pierden datos, por lo que no vale como solución.

La respuesta está en que los cambios realizados a las opciones del compilador para priorizar la velocidad por encima de todo han resultado ser contraproducentes en una ejecución real. Se ha conseguido aumentar la velocidad a costa de sobrescribir las variables en los registros, una encima de otra, cada vez que el micro iba a trabajar con ellas, sin ningún tipo de orden. El resultado ha sido una mejora en la velocidad pero unos valores de actuación que no tienen nada que ver con lo que debe salir, ya que a mitad de la ejecución se pierden o se sobrescriben.

Se ha decidido por tanto volver a las opciones de compilador antiguas, observándose en efecto que ahora que no se prioriza la velocidad sólo unos pocos datos se guardan en los registros (ya que no caben todos), y por tanto volviendo al problema inicial: no se llega a 100 kHz.

La pérdida de velocidad (prácticamente el 80 %) se debe a que el procesador trabaja con números en coma flotante, y cuyas multiplicaciones llevan un tiempo significativo. El problema de los modelos teóricos de control es que siempre tienen coeficientes con decimales, ya que se necesita una precisión para realizarlo correctamente. La única solución que queda entonces es trabajar en coma fija.

Por tanto se ha decidido que hay que trabajar con números enteros. Para ello, se necesita hacer un redimensionado de los coeficientes, es decir, multiplicarlos por una potencia de 2 lo suficientemente alta (por ejemplo, por 2^{10}= 1024) y seguidamente redondearlos al número entero más cercano, reduciendo así el error

por precisión a niveles mucho más aceptables y que no van a cambiar el resultado de forma significativa.

El problema de redimensionar los coeficientes es que la actuación ahora también va a salir redimensionada y en otra escala totalmente distinta a la inicial. Por partes, la lectura del dato del ADC puede dar entre 0 y 4096. La consigna se ha establecido es 1750 sobre 4096 (5 V la misma escala del ADC). Los coeficientes del modelo formal están escalados todos por 1024 (2^{10}).

Por lo tanto hay que hacer un re-escalado hacia abajo una vez que en la ecuación en diferencias se haya trabajado con enteros, que era el propósito principal de este problema. Por tanto, hay que desplazar a la derecha 10 posiciones el resultado y0 (la actuación), es decir, dividir por 2^{10}, para volver a tener y0 en una escala en la que sólo influye el error.

El último cambio que se necesita hacer es un cambio de escala de los coeficientes que multiplican al error, para llevarlos a la misma escala que la actuación. Para ello hay que dividir el coeficiente escalado por 4096 (escala del error), multiplicar por 12 voltios (que es el máximo posible que puede dar la salida) y finalmente para obtener el número de ciclos multiplicar el resultado por 1000 (que es un ciclo del 100 %). El resultado de estas operaciones se puede reducir a una sola, una multiplicación por un factor de 2,93 (ecuación 3).

Ecuación 3: $$Coef = b_k \cdot \frac{1000\ ciclos \cdot 12\ V}{4096} = b_k \cdot 2{,}93$$

Con estos cambios, se eliminan del programa de control las operaciones con números de coma flotante. Los resultados muestran una clara mejoría en el tiempo que se tarda en ejecutar el programa. La velocidad se ha incrementado de nuevo a niveles muy superiores a 100 kHz, por lo que se ha solucionado el problema de la velocidad, y ya no sólo eso, sino que se dispone de tiempo suficiente dentro de cada ejecución del programa para añadir operaciones extra si fuese necesario.

5.3 Segunda versión del regulador

5.3.1 Declaración del PWM doble complementario

Según se ha visto en el apartado 4.3, al principio del desarrollo de la aplicación se inicializaron varios módulos del micro, entre los que estaba el PWM. En aquel momento no se prestó demasiada atención a la configuración del mismo, y únicamente se limitó a ver que funcionaba correctamente, pero ahora que se ha

llegado a una primera versión completa, es necesario configurar de forma final el PWM (código 7).

```
1. EPwm2Regs.TBPRD = 1000;
2. EPwm2Regs.CMPA.half.CMPA = 0;
3. EPwm2Regs.CMPB = 0;
4. EPwm2Regs.TBCTR = 0;
// Registros que activan un PWM doble con dead band
5. EPwm2Regs.DBCTL.bit.IN_MODE = 0;
6. EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
7. EPwm2Regs.DBCTL.bit.OUT_MODE = 3;
8. EPwm2Regs.DBFED = 6;
9. EPwm2Regs.DBRED = 6;
```

Código 7: Configuración de un PWM doble con dead band para drivers

La regla de diseño principal que se ha seguido es la de desarrollar un PWM que genere dos señales simultáneamente: mientras una de ellas esté activa, la otra debe estar inactiva, y viceversa, es decir, se trata de dos señales de PWM complementarias entre sí. Esto es necesario para poder controlar el convertidor.

Observando el código del PWM, se puede ver cómo las 4 primeras líneas son iguales a la configuración inicial, simplemente se declara que el número de ciclos totales es de 1000 (100 kHz) y que los comparadores de subida y bajada se inicializan por el momento a 0. En las líneas 5 a 9 es donde se configura el PWM para que genere dos señales complementarias en dos puertos GPIO distintos, y cuya regla de generación es la de activo alto complementario (DB_ACTV_HIC) de tal forma que cuando una señal esté a 1, la otra se mantenga a 0.

Finalmente las líneas 8 y 9 indican los tiempos muertos que tiene que haber entre el cambio de una señal con su complementaria. Se ha dejado un valor de 6 que es el equivalente a 60 ns. En la hoja de especificaciones del convertidor conmutado se especifica un tiempo muerto entre señales de al menos 40 ns para que nunca haya riesgo de cortocircuito, por lo que 60 debería ser más que suficiente.

Para cerciorarse de que la configuración del PWM es la correcta, se ha procedido a hacer una prueba para verificar el correcto funcionamiento del PWM doble complementario. Las señales, utilizando el módulo PWM 2, se generan en los puertos GPIO2 y GPIO3 respectivamente. Para hacer esta primera prueba se ha dado un valor nominal fijo de 700 ciclos al PWM activo alto (figura 15).

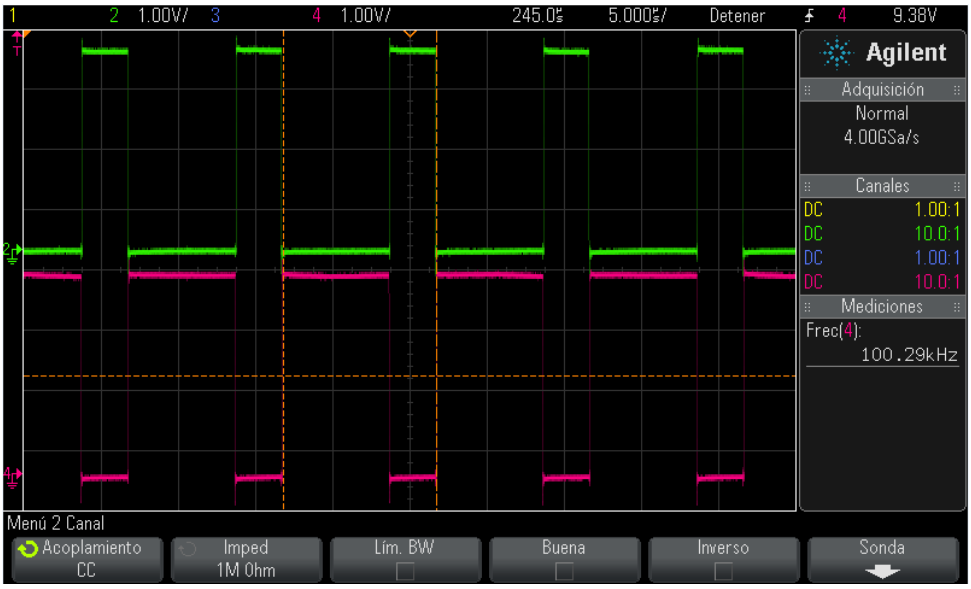


Figura 15: PWM doble complementario con dead band a 100KHz

Las pruebas realizadas han resultado ser satisfactorias, ya se genera un PWM con el ciclo de trabajo especificado y su complementario con el ciclo restante a una frecuencia de 100KHz. En la figura 16 se puede ver con mucho más detalle el cambio entre las dos señales de PWM, con el tiempo muerto realizado exactamente a 60 ns (como se especifica en [18] y [10]).

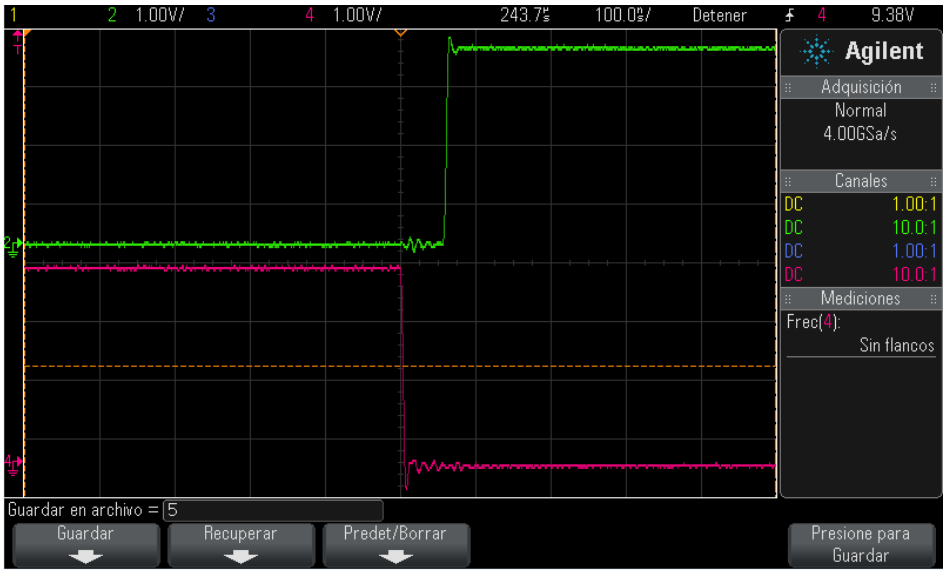


Figura 16: Detalle de la transición entre señales del PWM doble complementario

5.3.2 Construcción de un botón pull-down

Al desarrollar la aplicación de control, hay que tener en cuenta que el programa se ejecuta y empieza a leer datos inmediatamente después de que el Code Composer Studio lo ejecute. Eso quiere decir que desde el momento en el que se inicializa el programa hasta que se conecte el sistema completo con el regulador, se estarán leyendo datos y controlando cosas que no interesan. Es necesario entonces retrasar el proceso del regulador hasta que se le dé una señal.

Este proceso se ha realizado con un mecanismo sencillo de botón conectado a un puerto GPIO que funcionará como puerto de entrada. El botón cuenta con una conexión con una resistencia pequeña a VCC (3,3 V) y una más grande a GND, de tal forma que cuando el botón esté sin pulsar, el puerto GPIO leerá un '0', y si lo pulsamos leerá un '1'. En la figura 17 se puede ver el montaje final del botón.

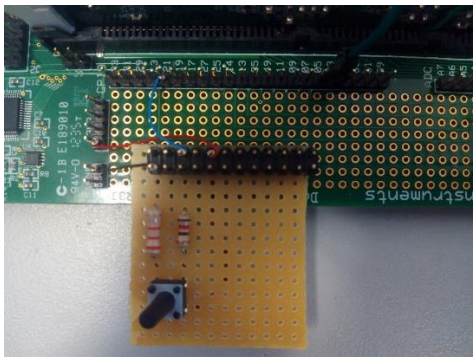


Figura 17: Botón que actúa como interruptor en un puerto GPIO

Para que el programa espere al botón, se ha declarado el puerto GPIO 21 como un puerto de entrada, con pull-down y que únicamente actúe como un GPIO de entrada normal (código 8).

```
1. GpioCtrlRegs.GPADIR.bit.GPIO21 = 0;
2. GpioCtrlRegs.GPAPUD.bit.GPIO21 = 1;
3. GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;
4. GpioCtrlRegs.GPAQSEL2.bit.GPIO21 = 3;
```

Código 8: Puerto GPIO destinado a la entrada de un botón en la placa

De la misma forma ya en la propia ejecución del programa se ha programado un sencillo bucle while que espera a que haya un '1' en el puerto GPIO destinado al botón antes de comenzar a leer y hacer el control. Mientras no se pulse el botón el programa se queda en un estado de espera infinito (código 9).

```
// Bucle que espera a un interruptor
1. while (GpioDataRegs.GPADAT.bit.GPIO21 == 0) {}
```

Código 9: Bucle que espera a que se pulse un botón

5.4 Versión final del regulador

En el apartado 5.1 se realizó el programa de control en un bucle infinito que realizaba todas las acciones: leer del ADC, hacer la ecuación en diferencias y pasar el ciclo al PWM. En ese momento (código 5, línea 23) se utilizó un bucle for (para que el bucle general del programa fuese a 100 kHz) que introducía un delay realizado "a mano", por lo que no se ha alcanzado a obtener una precisión de 100 kHz exactos, lo que ha ocasionado que el PWM y el bucle que realiza todas las operaciones no estén bien sincronizados entre sí. Este problema provoca que algunas muestras del ADC o del ciclo de trabajo no se capturen correctamente.

Para resolver el problema, es necesario hacer que tanto el PWM como la ejecución del programa de control vayan sincronizadas exactamente a 100kHz, por lo que es necesario utilizar el timer de la CPU. En este caso, lo que se ha hecho es mantener en el bucle principal del main (bucle infinito) las lecturas del ADC (código 10, líneas 1 a 6), mientras que todo el algoritmo de control, en el que se incluye la ecuación en diferencias, va a ir en la función del timer (código 10, líneas 7 a 27), que está declarado para que se ejecute exactamente cada 10 µs, por lo que ahora sí se trabajará exactamente a 100 kHz sin necesidad de ningún delay.

```
1. for(;;)
2. {
3.       while (AdcRegs.ADCST.bit.INT_SEQ1== 0) {}
4.       AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;
5.       leidoADC1 = ((AdcRegs.ADCRESULT0>>4));
6. }

7. interrupt void cpu_timer0_isr(void)
8. {
9.      // GPIO que se pone a 1 al comenzar la ejecución
10.      GpioDataRegs.GPADAT.bit.GPIO20= 1;
11.      e0=consignaPrueba-leidoADC1;
12.      // Ecuacion en diferencias
13.      y0 =((b0*x0)+(b1*x1)+(b2*x2))-((a1*y1)+(a2*y2));
```

```
14.      // Se debe volver a quitar el escalado de 2^10
15.      y0 = y0 >> 10;
16.      // Saturación de máximo y mínimo
17.      if (y0 > 1000) y0 = 1000;
18.      else if (y0 < 0) y0 = 0;
19.      // Desplazamos en el tiempo
20.      y2=y1; y1=y0; x2=x1; x1=x0;
21.      // Ciclo de trabajo al PWM
22.      EPwm2Regs.CMPB = ciclo;
23.      // GPIO que se pone a 0 al terminar la ejecución
24.      GpioDataRegs.GPADAT.bit.GPIO20= 0;
25.      // Asentir la interrupción del timer
26.      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
27.      }
```

Código 10: Función del timer que hace el algoritmo de control

Es necesario asentir la interrupción al final (línea 26) para que pueda volver a ejecutarse el timer pasados 10 µs. Uno de los cambios con respecto a la primera versión es la línea 9, donde se puede ver como la actuación se reescala de nuevo por 2^{10}. El resto de líneas tienen la misma función que en el código de programa explicado anteriormente (código 5).

5.4.1 Guardado de variables para la ejecución

Finalmente se ha realizado una pequeña adición en la aplicación de control que se realiza en el timer. Para poder realizar un mejor seguimiento del funcionamiento del programa, se han guardado las 200 primeras muestras de la actuación y de los valores leídos por el ADC en un vector, dando la oportunidad de ver a posteriori todos los valores que han ido tomando. Esto facilita la depuración en el caso de que hubiese algún problema inesperado, identificando rápidamente si el PWM o el ADC tienen algún valor incorrecto.

6. CONEXIÓN DEL SISTEMA COMPLETO

Una vez desarrollada la aplicación de control en el microcontrolador y vistas las especificaciones generales del convertidor conmutado, es hora de conectar el sistema completo y dejarlo preparado para ejecutar modelos de control de forma final [19]. Recapitulando, las señales del microcontrolador son las siguientes:

- El PWM generado por el µC sale por dos pines distintos, el GPIO 2 y GPIO 3. El primero irá conectado al MOSFET HSM que actúa como interruptor 1, mientras que el segundo irá conectado al MOSFET LSM (interruptor 2). En el regulador estos pines están señalados como H1 y L1 (high 1 y low 1). Dado que los demás MOSFET no se van a utilizar en las pruebas con una sola fase, se han dejado todos los restantes conectados a GND.

- El puerto GPIO 20 del micro indica la velocidad con la que se está ejecutando el programa de control. No es necesario conectarlo a ningún sitio en las pruebas finales, simplemente sirve para ver la velocidad en caso de que se quiera depurar.

- El puerto GPIO 21 está conectado al botón de inicio que se explicó en la sección 5.3.2.

- Para asegurar el correcto funcionamiento de los pines conectados entre el micro y el regulador conmutado, se han conectado las masas de ambos.

Las conexiones realizadas en el regulador son:

- Se ha utilizado una fuente de alimentación moderna que soporta intensidades del orden de 6 A. Se ha utilizado el modelo Power Supply EA-PS 2342-06 B que permite hacer un preset con los voltajes y las intensidades máximas que va a soportar cada uno de los puertos.

- En las especificaciones del convertidor se detalla que existen dos alimentaciones distintas, una para la placa y otra para los drivers. Para la alimentación de la placa se utiliza una estándar de 12 V (alimentación analógica). Para los drivers MOSFET se ha consultado en la hoja de datos correspondiente al modelo construido [18]. Se ha visto que pueden trabajar con valores de entre 10 y 20 V, por lo que se optó por dejar también a 12 V la alimentación digital.

- Se ha conectado la tabla de resistencias en el conector de salida de voltaje (la salida del regulador sin modificar). Dado que las resistencias no tienen polaridad, no es necesario mirar en qué pin exactamente debe ir conectada la entrada y salida de las resistencias.

- El pin de salida del divisor de tensión se conecta directamente al pin A0 del microcontrolador. Este pin será el que el ADC del micro lea en cada iteración.

De forma esquemática, la conexión del sistema completo queda de la siguiente forma (figura 18).

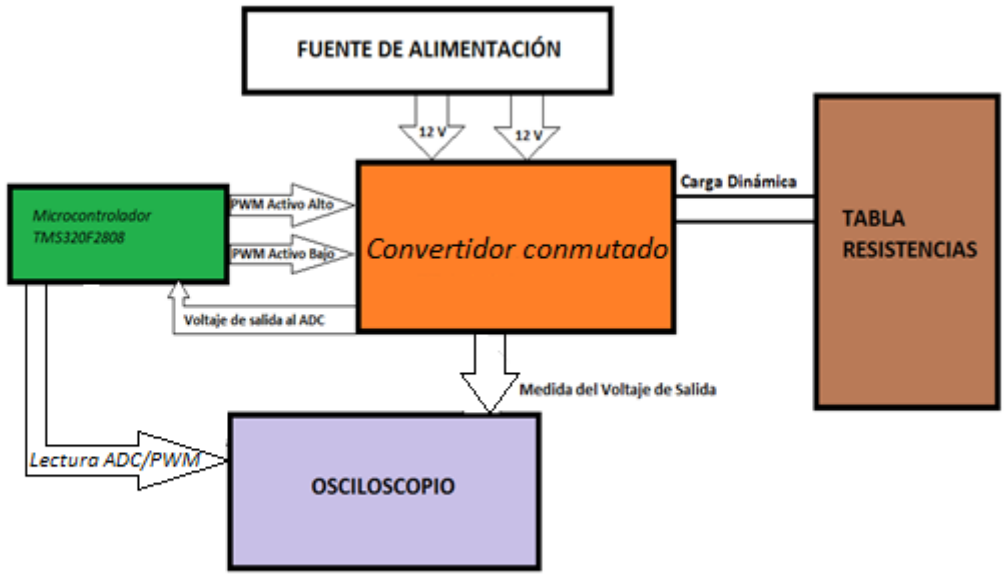


Figura 18: Esquemático completo del sistema con las conexiones pertinentes

En el osciloscopio se utilizan tres sondas que se conectan, respectivamente, al pin activo alto del PWM del micro, al puerto GPIO 20 para medir la velocidad y finalmente la tercera a la salida del sistema tras el divisor de tensión.

Finalmente se ha medido, de forma experimental, el valor que tiene el condensador y la bobina del regulador, para asegurar que los valores teóricos que se especificaban eran correctos. Las medidas tomadas son de 22 µH para la bobina (valor correcto), y aproximadamente 190-200 µF en el condensador, así pues, esta pequeña variación entre el valor real y el teórico en el condensador puede que afecte más tarde a las pruebas con modelos de segundo orden.

El sistema completo ya montado para las pruebas con los modelos finales (figura 19).

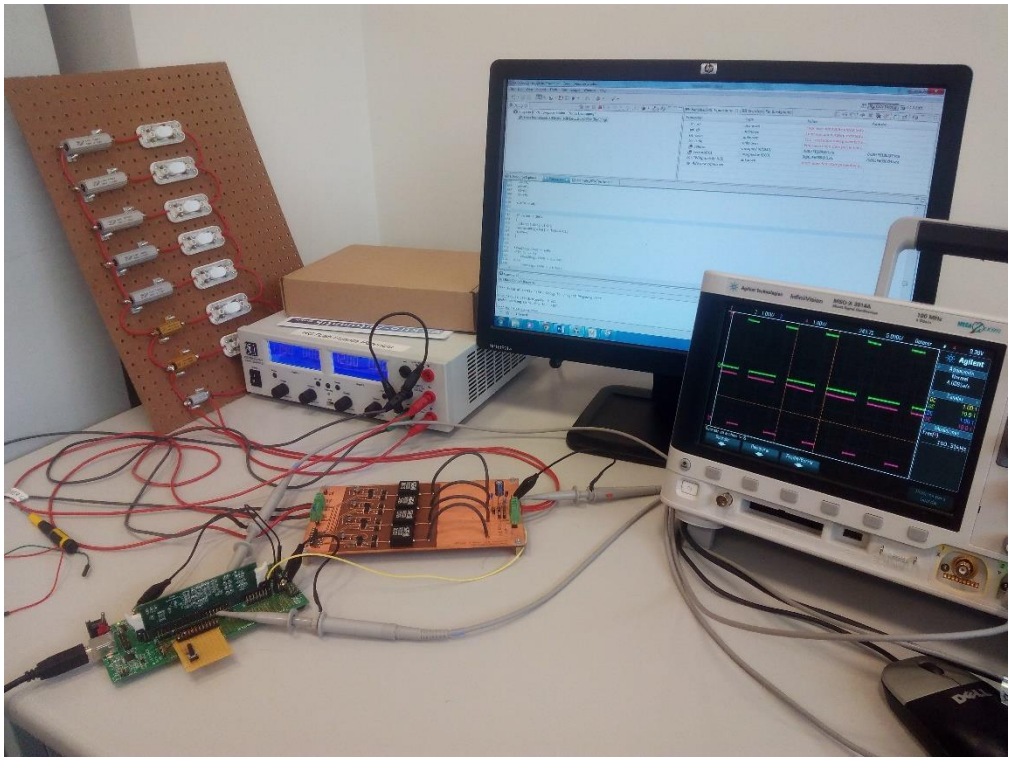


Figura 19: Conexión del sistema completo para las pruebas finales

7. PRUEBAS EXPERIMENTALES

En este capítulo se van a exponer las diferentes pruebas que se han realizado con la conexión del sistema completo. Se van a realizar primero unas pruebas de sincronización y de respuesta al escalón para diferentes cargas y posteriormente se probarán dos reguladores (de primer y segundo orden) para ver el comportamiento del regulador.

7.1 Sincronización de Timer y PWM

Antes de realizar las pruebas con modelos formales es necesario asegurarse de que el timer y el PWM doble complementario funcionan bien. Hasta ahora únicamente se han realizado las pruebas por separado para asegurarnos de que se estaba realizando el algoritmo de control a 100 kHz, por lo que hay que comprobar si además de ir a la misma frecuencia, también están sincronizados entre ellos.

El resultado de las medidas ha sido positivo: tras declarar el PWM se ha iniciado el timer y mediante un pequeño delay en el PWM, ambas señales quedan perfectamente sincronizadas (figura 20).

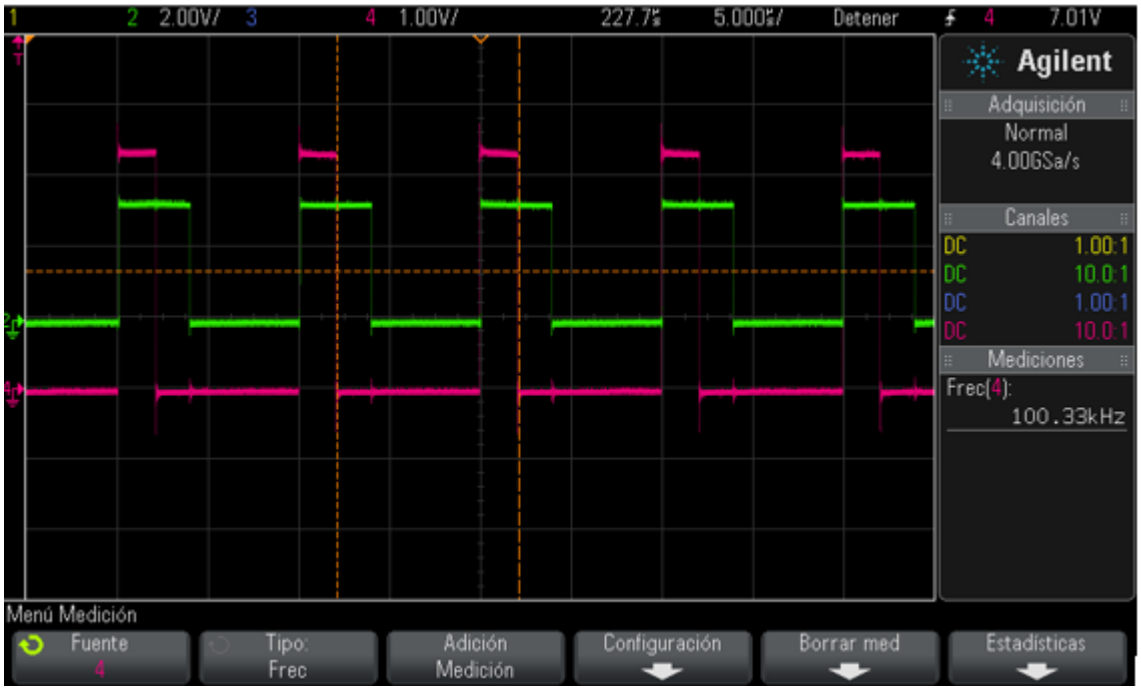
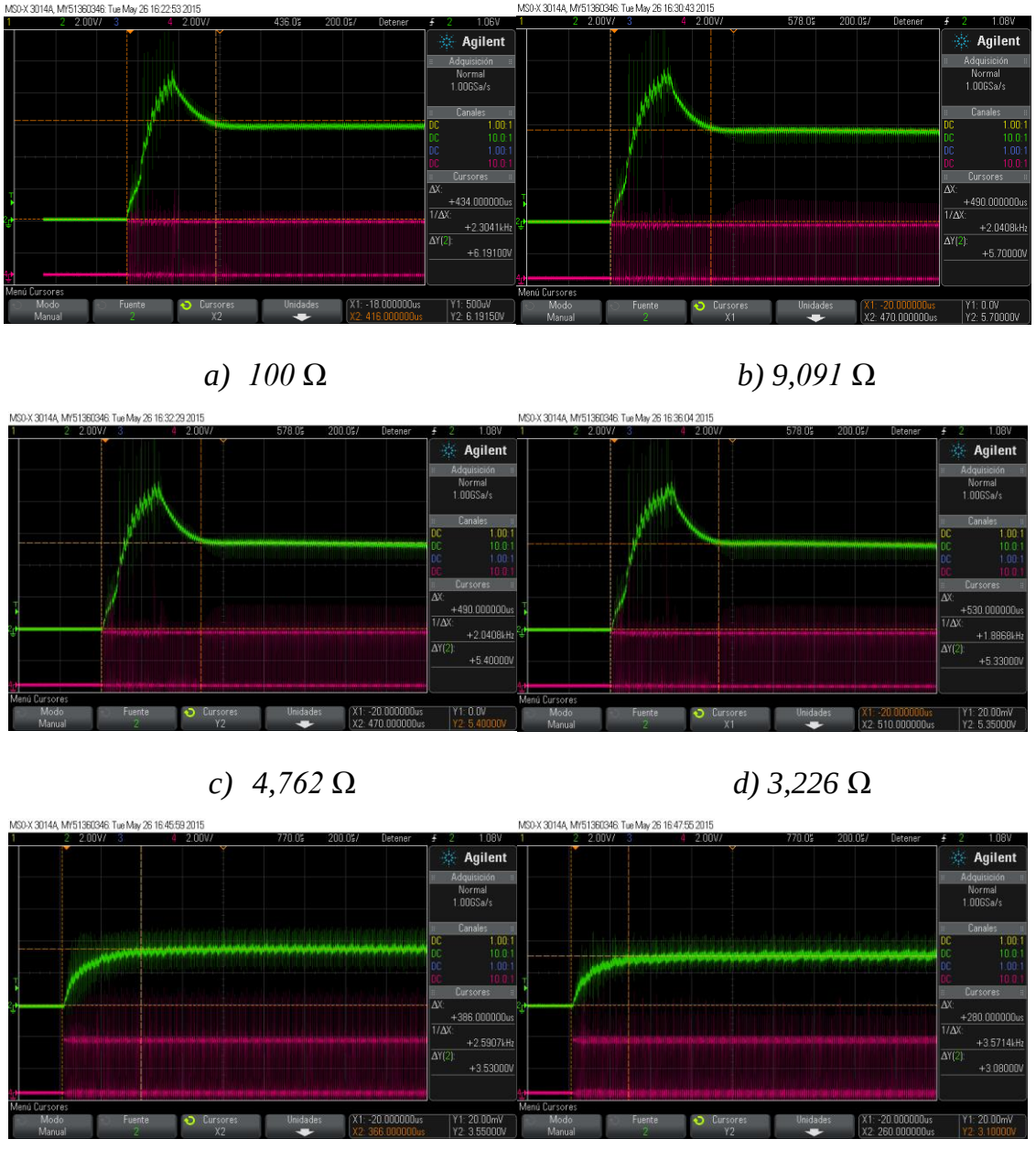


Figura 20: Pruebas de sincronización entre el PWM (canal 2) y el timer de la CPU (canal 4).

7.2 Respuesta al escalón para diferentes cargas en lazo abierto

Una vez comprobado el correcto funcionamiento del PWM, se ha procedido a mostrar la respuesta al escalón para diferentes cargas (cambiando la resistencia equivalente en la tabla de resistencias), para un ciclo de trabajo fijo del 50% (figura 21).



a) 100 Ω *b) 9,091 Ω*

c) 4,762 Ω *d) 3,226 Ω*

e) 1,639 Ω *f) 0,474 Ω*

Figura 21: Respuesta al escalón en lazo abierto para diferentes cargas

7.3 Regulador de primer orden

Se han propuesto para hacer las pruebas experimentales tres reguladores distintos, de primer orden y segundo orden, para que se puedan apreciar las diferencias del control cuando aplicamos una regulación de ambos tipos.

El regulador de primer orden desarrollado se puede ver en la ecuación 4:

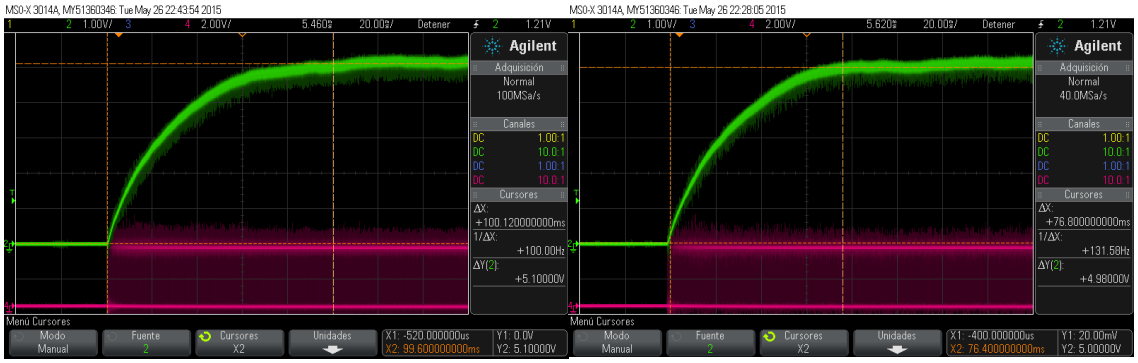
Ecuación 4: $$R(z) = 0{,}00002 \cdot \frac{(z+0{,}84)}{z-1}$$

La ganancia del regulador es baja, por lo que se espera que la dinámica transitoria del regulador sea lenta. Para poder trasladar los coeficientes al programa desarrollado en el trabajo se necesita escalarlos (tabla 2). Esta escala queda así (el valor final es el producto multiplicado por el coeficiente 2,93 (pág. 30) redondeado al entero más próximo, pero sólo aplica para los coeficientes b).

Coeficientes	**Valor núm.**	**Potencia**	**Producto**	**Redondeo**	**Valor final**
b1	0,00002	18	5,24	5	15
b2	0,0000168	18	4,4040	4	12
a1	1	18	262144	262144	262144
a2	-1	18	262144	262144	262144

Tabla 2: Transformación de coeficientes del modelo de primer orden

La figura 22 muestra los resultados experimentales que se han probado con el regulador de primer orden para una carga de 100 Ω y posteriormente con el primer interruptor de las resistencias activado (9,091 Ω) para un arranque de la aplicación desde cero:



a) 100 Ω *b) 9,091 Ω*

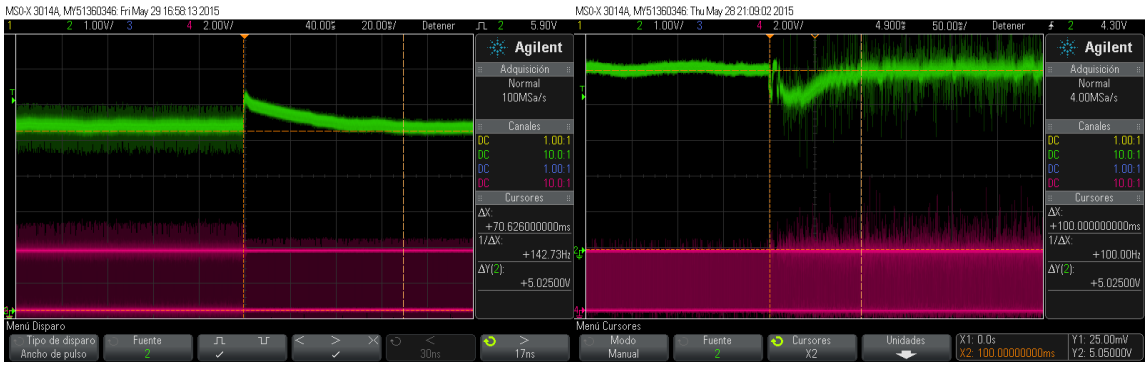
Figura 22: Pruebas de arrancado con regulador de primer orden con 100 Ω y 9,091 Ω

Ahora se han realizado pruebas de arranque pero esta vez desde 1 V hasta 5 V, es decir, con la aplicación ya encendida, también para ambas cargas (figura 23).

a) 100 Ω *b) 9,091 Ω*

Figura 23: Escalón de 1 a 5 V (regulador de primer orden)

En las pruebas realizadas se puede ver que en las medidas realizadas en la práctica, con una carga mayor el tiempo que pasa hasta que la dinámica se estabiliza a 5 voltios es siempre algo menor que el caso de 100 Ω. Finalmente, el escalón de carga (de 100 Ω a 1,961 Ω y viceversa) para este regulador se puede ver en la figura 24.



a) 1,639 a 100 Ω *b) 100 a 1,639 Ω*

Figura 24: Escalones de carga del regulador de primer orden

7.4 Regulador de segundo orden

Para el regulador de segundo orden se ha desarrollado un modelo que vaya mucho más rápido que el modelo de primer orden, para apreciar la variación de la dinámica con respecto al primero de forma más clara.

El modelo de regulador de segundo orden queda (ecuación 5):

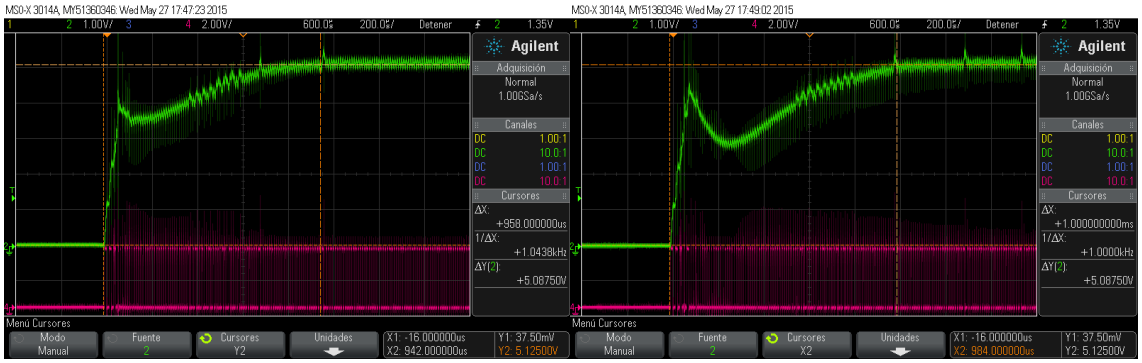
Ecuación 5 $$R(z) = 0{,}25 \cdot \frac{(z^2 - 1{,}87z + 0{,}882)}{(z-1)\cdot z}$$

En este caso la ganancia es mucho más alta, por lo que el transitorio hasta que el modelo se estabilice en 5 V será diferente a los anteriores. Los coeficientes necesitan ser escalados para poder trabajar con ellos en el programa. La conversión se puede consultar en la tabla 3.

Coeficientes	**Valor núm.**	**Potencia**	**Producto**	**Redondeo**	**Valor final**
b0	0,25	10	256	256	750
b1	-0,4675	10	-478,72	-479	-1403
b2	0,2205	10	225,79	226	662
a1	-1	10	-1024	-1024	-1024
a2	0	10	0	0	0

Tabla 3: Transformación de coeficientes del modelo de segundo orden

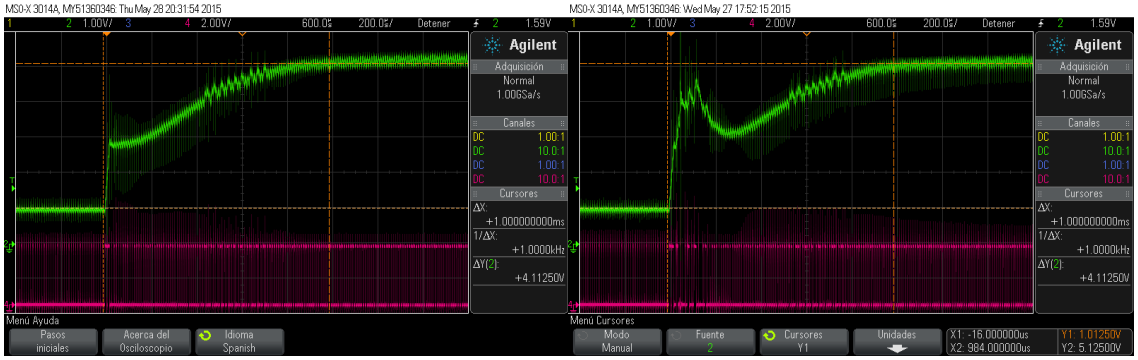
Aplicando los valores obtenidos se ha probado el modelo con pruebas experimentales. La figura 25 muestra la dinámica que sigue el control desde un arranque desde cero hasta que se estabiliza a 5 V.



a) 100 Ω *b) 9,091 Ω*

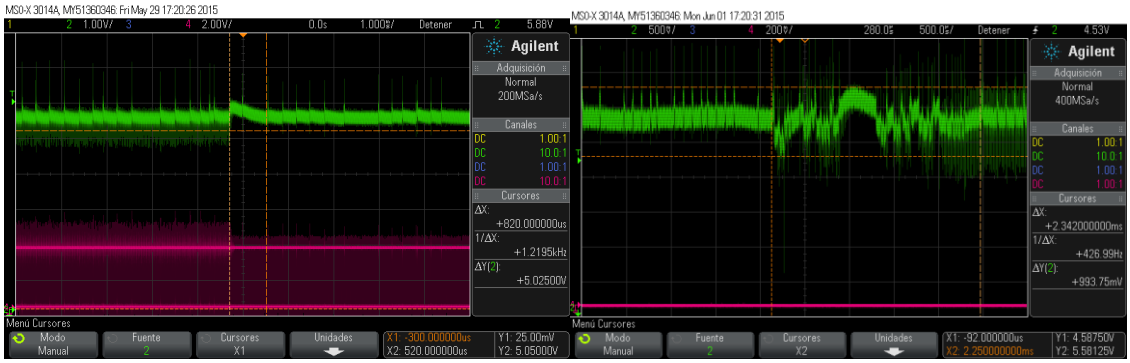
Figura 25: Escalón de 0 a 5 V (regulador de segundo orden)

De la misma forma, se han tomado medidas para ver si la dinámica se mantiene cuando aplicamos el control bajo una situación de 1 V (en vez de arrancar desde 0). El resultado está en la figura 26. Posteriormente (figura 27) se han probado los escalones de carga cuando se cambia la resistencia para ver cuánto tarda en estabilizar el sistema de nuevo a 5 V.



a) 100 Ω *b) 9,091 Ω*

Figura 26: Escalón de 1 a 5 V (regulador de segundo orden)



a) 4,762 a 100 Ω *b) 100 a 4,762 Ω*

Figura 27: Escalones de carga del regulador de segundo orden

En el segundo caso la dinámica es distinta debido al ruido que mete en ese instante el interruptor, aunque se estabiliza a 5 V de igual forma.

7.5 Regulador de segundo orden (alta ganancia)

Finalmente se ha probado un regulador de segundo orden con una ganancia más alta que hace que la dinámica sea aproximadamente el doble de rápida con respecto al segundo regulador probado.

La ecuación 6 muestra el modelo del regulador utilizado:

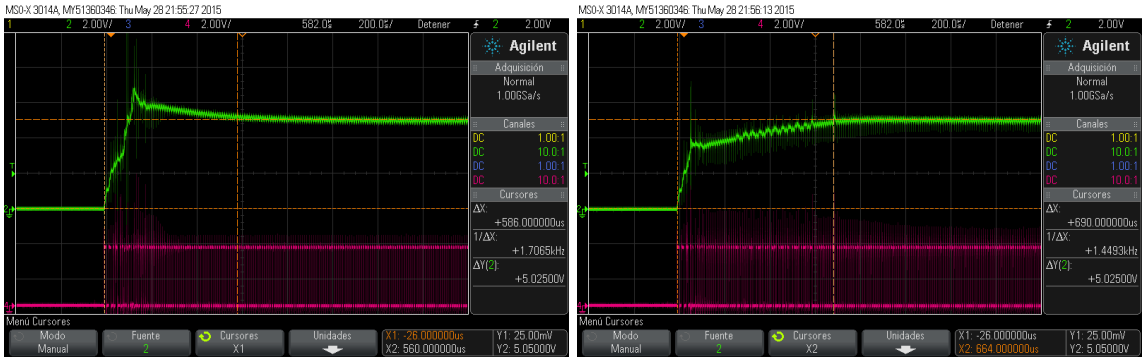
Ecuación 6 $$R(z) = 0{,}5 \cdot \frac{z^2 - 1{,}89z + 1{,}989}{(z-1)\cdot z}$$

Se ha doblado la ganancia con respecto al segundo regulador, además de modificarse las posiciones de los ceros. Los coeficientes transformados al programa se pueden ver en la tabla 4.

Coeficientes	**Valor núm.**	**Potencia**	**Producto**	**Redondeo**	**Valor final**
b0	0,5	10	512	512	1500
b1	-0,99	10	-1013,76	-1014	-2970
b2	0,4945	10	506,368	506	1483
a1	-1	10	-1024	-1024	-1024
a2	0	10	0	0	0

Tabla 4: Transformación de coeficientes del modelo de segundo orden (alta ganancia)

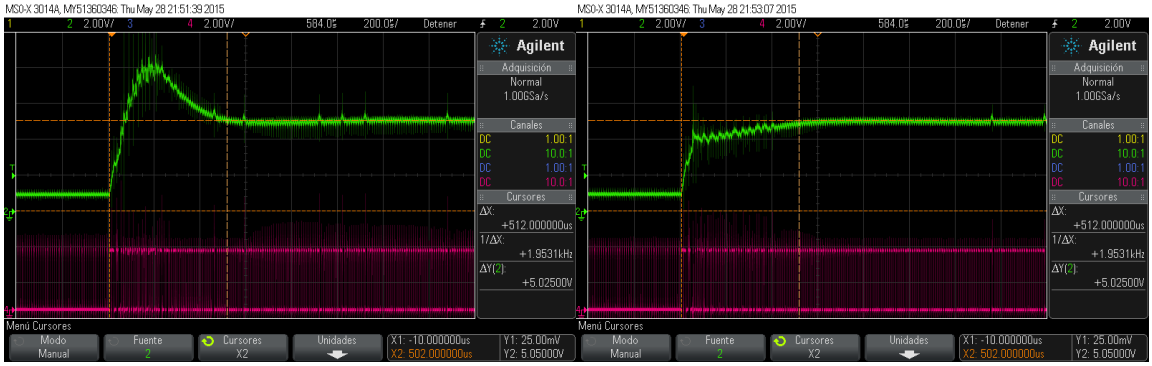
En la figura 28 se puede observar el escalón de arranque para los casos de 100 Ω y 9,091 Ω. Se puede observar que en el primer caso existe una pequeña sobreoscilación antes de estabilizarse en 5 V.



a) 100 Ω *b) 9,091 Ω*

Figura 28: Escalón de 0 a 5 V (regulador de segundo orden de alta ganancia)

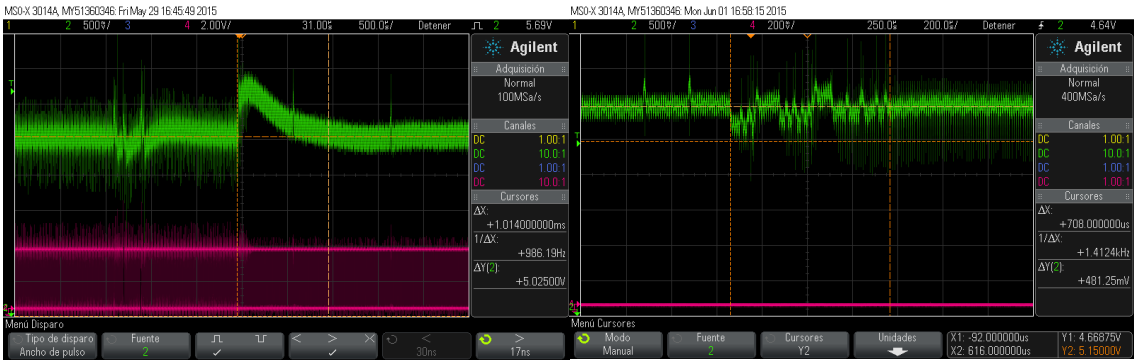
Se ha probado seguidamente el escalón de 1 a 5 V (figura 29), observándose un sobreoscilamiento algo más claro en la prueba con 100 Ω. Se puede observar la inestabilidad que provoca que haya baja carga en un regulador de alta ganancia.



a) 100 Ω *b) 9,091 Ω*

Figura 29: Escalón de 1 a 5 V (regulador de segundo orden de alta ganancia)

Los escalones de carga (100 Ω a 1,639 Ω y viceversa) de este regulador están en la figura 30.



a) 1,639 a 100 Ω *b) 100 a 1,639 Ω*

Figura 30: Escalones de carga (regulador de segundo orden de alta ganancia)

El ruido del interruptor hace que el escalón de carga no se observe bien en el segundo caso (figura 30 b). Aun así, el regulador mantiene la tensión de salida correctamente.

8. CONCLUSIONES

En este trabajo de fin de grado se ha realizado una aplicación de control en un microcontrolador para regular un convertidor conmutado. La primera parte del trabajo consistió en desarrollar en Code Composer Studio un programa que sirviese para realizar una aplicación de control digital. Para ello, se configuraron los módulos necesarios del µC: el ADC, el PWM y el timer.

Una vez realizada una primera versión del programa se encontró un problema en la velocidad de procesamiento que se solucionó, tras varias pruebas, trabajando con números enteros (coma fija) mediante un escalado de los coeficientes del regulador. Para resolver además el problema de la sincronización entre el PWM y el programa, se necesitó utilizar el timer disponible de la CPU.

Cuando se terminó de desarrollar el código de la aplicación de control, se procedió a hacer pruebas experimentales de la respuesta al escalón para diferentes cargas en lazo abierto. Finalmente se probaron tres reguladores diferentes para ver cómo se comportaban en pruebas prácticas. Los distintos reguladores han cumplido su tarea manteniendo el valor de la tensión de salida.

8.1 Trabajo futuro

Para finalizar, hay que comentar algunos cambios de cara a mejorar este trabajo en un futuro. El primer cambio que se podría hacer es trabajar a una velocidad mayor, si el micro o DSP lo permite, y el conmutador lo soporta. Una velocidad de 100 kHz está bien pero en el mercado se pueden encontrar aplicaciones más rápidas en este momento.

Otra posible mejora es la de utilizar el convertidor completo multifase, generando cuatro señales de PWM desfasadas 90º una con respecto a la siguiente. Finalmente, se puede probar la aplicación de control desarrollada en un convertidor conmutado reductor de uso comercial para ver las diferencias entre el utilizado y el comprado.

BIBLIOGRAFÍA

1. N. Daniel Trip, S. Dale, V. Popescu, "Digital Control for DC/DC Switched Mode Converters", "http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber= 5679257&tag=1" Noviembre 2010.

2. Tzu-Ching Chia, Chun-Lin Lu, "Design and Implementation of the Microcontroller Control System for Vertical-garden Applications", 2011 Fifth International Conference on Genetic and Evolutionary Computing, NSC 99-2218-E-168-001.

3. Ms. Lei Lei Yin Win, Ms. Phyu Hnin Khaing , Dr. Mg Mg Latt, "Design Considerations for Microcontroller Based Process Control for Washing Machine", 2009, "http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber =5190505".

4. S. Huseinbegovic, S. Kreso, O. Tanovic, "Development of a Distributed Elevator Control System Based on the Microcontroller PIC 18F458", IEEE Region 8 SIBIRCON, Julio 2010, "http://ieeexplore.ieee.org/stamp/stamp.jsp?tp= &arnumber=5555091".

5. D. Baba, Texas Instruments, "Benefits of a multiphase buck converter", 2012 "http://www.ti.com/lit/an/slyt449/slyt449.pdf".

6. J. Leppaaho, T. Suntio, "Characterizing the Dynamics of the Peak-Current-Mode-Controlled Buck-Power-Stage Converter in Photovoltaic Applications", Julio 2014, "http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber =6590001".

7. T.-F. Wu, S.-Y. Tseng, J.-S. Hu and Y.-M. Chen, "Buck and Boost Derived Converter for Livestock/Poultry Stunning Applications", 2006, "http://ieeexplore .ieee.org/stamp/stamp.jsp?tp=&arnumber=1620743".

8. M. B. Camara, B. Dakyo, H. Gualous, "Energy Management base don Two-phase interleaved Buck-boost and Boost converters for Electric Vehicles Applications – Using Lithium-battery and Fuel cell", 2011, "http://ieeexplore .ieee.org/stamp/stamp.jsp?tp=&arnumber=6477433".

9. R. W. Erickson, D. Maksimovic, "Fundamentals of Power Electronics" Segunda Edición, Kluwer Academics Publishers, 2004 ISBN 0-7923-7270-0.

10. L. Usero, "Diseño, Implementación y Control de Reductor Multifase", TFG Escuela Politécnica Superior, Universidad Autónoma de Madrid, Septiembre 2014.

11. Texas Instruments, “TMS320F2808 Data Manual”, 2003, “http://www.ti.com/lit/ds/symlink/tms320f2801.pdf”.

12. Mathworks, “Introduction: Digital controller Design”, “http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlDigital”.

13. J. Martínez García-Tenorio, “Digital Control Techniques for DC/DC Power Converters”, Tesis Doctoral, Helsinki University of Technology, Agosto 2009.

14. K. Schachter, “TMS320F2808 Peripheral Programming”, “http://www.ti.com/ lit/ml/sprp625/sprp625.pdf”, Marzo 2007.

15. Texas Instruments, “Enhanced Pulse Width Modulator (ePWM) Guide”, “http://www.ti.com/lit/ug/sprug04a/sprug04a.pdf”, Octubre 2008, revisado Julio 2009.

16. Texas Instruments, “TMS320 DSP Algorythm Standard”, Junio 2005, revisado Febrero 2007. “http://www.ti.com/lit/ug/spru352g/spru352g.pdf”.

17. Texas Instruments, “Loop Stability Analysis of Voltage Mode Buck Regulator With Different Output Capacitor Types – Continuous and Discontinuous Modes”, Abril 2008, “http://www.ti.com/lit/an/slva301/slva301.pdf”.

18. International Rectifier, “IRF7855Pb Data Sheet”, Mayo 2006. “http://www.irf.com/product-info/datasheets/data/irf7855pbf.pdf”.

19. O. García, P. Zumel, A. de Castro, “Automotive DC–DC Bidirectional Converter Made With Many Interleaved Buck Stages”, Mayo 2006, IEEE 0885-8993.

APÉNDICE

Anexo 1: Abreviaturas

ADC – Analog to Digital Converter

Conversor DC/DC – Conversor de corriente continua a corriente continua

CPU – Central Processing Unit

DAC - Digital to Analog Converter

FPGA – Field Programmable Gate Array

GPIO – General Purpose Input/Output

HDL – Hardware Description Language

HSM – High Side MOSFET

LSM – Low Side MOSFET

MOSFET – Metal-Oxide-Semiconductor Field-Effect Transistor

PWM – Power-Width Modulation

SARAM - Sequential Access and Random Access Memory

TFG – Trabajo de Fin de Grado

USB – Universal Serial Bus

Anexo 2: Código del programa de control

```
#include "DSP280x_Device.h"
#include "DSP280x_Examples.h"
#include "DSP280x_EPwm_defines.h"
#include "math.h"

// Declaración de la funcion de interrupción del timer
interrupt void cpu_timer0_isr(void);
#define ADC_CKPS          0x1
#define ADC_SHCLK         0xf

//Variables de pruebas y depuración
Uint16 PWMprueba[5];
Uint16 cicloA = 750;
Uint16 cicloB = 250;

//Variables del programa
Uint16 valores[200];
Uint16 vectorADC1[200];
int cien=0;
int uno=0;
int repetir;

int delay; //variable solo para retraso de programa
Uint16 i=0;
int ciclo=0;

// Variables del ADC
Uint16 leidoADC1 = 0;
int leido = 0;

// Parámetros a cambiar manualmente según modelo MATLAB
// b0, b1 y b2 * 2^10 * 1000 * 12 / 4096 y redondeados al entero
int x0=750;
int x1=-1485;
int x2=742;
// a1 y a2 son: coef de MatLab * 2^10
int y1=-1024;
```

```
int y2=0;

//Vectores de error y actuacion en variables
long long a0=0;
long long a1=0;
long long a2=0;
long e0=0;
long e1=0;
long e2=0;

int consignaPrueba = 1740; // En una escala de 4096

int c = 0, d = 0, f=0; // variables de bucles
void main()

{

	// Inicializar todas las variables del sistema
	InitSysCtrl();

	// No dividir el reloj del Sistema por ningún factor (100 MHz)
	EALLOW;
	SysCtrlRegs.HISPCP.all = 0x0;
	EDIS;

	// Desactivar las interrupciones hasta que no estén todos los módulos
	inicializados
	DINT;

	// Inicializar los registros de control del programa
	InitPieCtrl();

	// Limpiar todos los flags de las interrupciones:
	IER = 0x0000;
	IFR = 0x0000;
	// Inicializar la tabla de interrupciones
	InitPieVectTable();

	// Inicializar los módulos del micro (PWM y ADC)
	InitAdc();
```

```
InitEPwm1Gpio(); // PWM para pruebas
InitEPwm2Gpio(); // PWM para Control MOSFET
// Registrar en la tabla de interrupciones cpu_timer_interrupt
EALLOW;
PieVectTable.TINT0 = &cpu_timer0_isr;
EDIS;

// Activar las interrupciones del ADC y del timer
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //timer interrupt
IER |= M_INT1; // Enable CPU Interrupt 1
EINT;
ERTM;

// Configuración del ADC
AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK;
AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS;
AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;      AdcRegs.ADCTRL1.bit.CONT_RUN
= 1;
AdcRegs.ADCTRL2.all = 0x2000; //Empezar a hacer la conversión




//GPIO que cambiará cuando termine de procesarse cada iteración
EALLOW;
GpioCtrlRegs.GPADIR.bit.GPIO20 = 1;
GpioCtrlRegs.GPAPUD.bit.GPIO20 = 0;
GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0;

//GPIO pull-down para el botón
GpioCtrlRegs.GPADIR.bit.GPIO21 = 0;
GpioCtrlRegs.GPAPUD.bit.GPIO21 = 1;
GpioCtrlRegs.GPAMUX2.bit.GPIO21 = 0;
GpioCtrlRegs.GPAQSEL2.bit.GPIO21 = 3;
EDIS;

// Bucle que espera a que se pulse el botón
while (GpioDataRegs.GPADAT.bit.GPIO21 == 0) {}
```

```
while (GpioDataRegs.GPADAT.bit.GPIO21 == 1) {}
while (GpioDataRegs.GPADAT.bit.GPIO21 == 0) {}
//PWM 2: Controla los drivers del conmutador
EPwm2Regs.TBPRD = 1000; // 1000 ciclos
EPwm2Regs.CMPA.half.CMPA = 0; // Iniciar comparadores
EPwm2Regs.CMPB = 0;
EPwm2Regs.TBCTR = 0;
// Controla la dead band
EPwm2Regs.DBCTL.bit.IN_MODE = 0;
EPwm2Regs.DBCTL.bit.POLSEL = 2;
EPwm2Regs.DBCTL.bit.OUT_MODE = 3;
EPwm2Regs.DBFED = 6;
EPwm2Regs.DBRED = 6;
// Inicialización de otros registros de control
EPwm2Regs.TBCTL.bit.CTRMODE = TB_DOWN;
EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;
EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;
EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;
EPwm2Regs.AQCTLA.bit.CBU = AQ_CLEAR;
// Pequeño delay para sincronizar timer y PWM
for(f=0;f<44;f++) {}

// Inicialización del timer:
// 100MHz CPU Freq, 10 us Period
InitCpuTimers();
ConfigCpuTimer(&CpuTimer0, 100, 10);
StartCpuTimer0();

// Bucle que dura infinitamente que lee las muestras del ADC y provoca
interrupción al acabar de leer cada muestra
for(;;)
{
       while (AdcRegs.ADCST.bit.INT_SEQ1== 0) {}
       AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // Asentimiento
       leidoADC1 = ((AdcRegs.ADCRESULT0>>4));
```

```
	}
}
interrupt void cpu_timer0_isr(void)
{
	// Se cambia el valor de GPIO al empezar y al terminar
	GpioDataRegs.GPADAT.bit.GPIO20= 1;
	//Generación de la señal de error
	leido = (int) leidoADC1;
	e0=consignaPrueba-leido;

	// Ecuacion en diferencias
	a0 =((x0*e0)+(x1*e1)+(x2*e2))-((y1*a1)+(y2*a2));
	// Ciclo de trabajo actual se re-escala en 2^10
	a0 = a0 >> 10;
	// Saturación de máximo y mínimo
	if (a0 > 1000) a0 = 1000;
	else if (a0 < 0) a0 = 0;

	// Desplazamos en el tiempo
	a2=a1; a1=a0; e2=e1; e1=e0;

	// ciclo de trabajo
	ciclo = a0;

	// 200 primeros valores para depurar
	if (cien < 200)
	{
		valores[cien]=ciclo;
		vectorADC1[cien] = leidoADC1;
		cien++;
	}
	// Se le pasa el valor al módulo PWM
	EPwm2Regs.CMPB = ciclo;

	// Se le pasa el valor al módulo PWM
	GpioDataRegs.GPADAT.bit.GPIO20= 0;
	// Asentir la interrupción para que puedan llegar más
	PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```